NASA CR-73348
Available to the Public

FINAL REPORT

MULTIPAC, A MULTIPLE POOL PROCESSOR AND COMPUTER FOR A SPACECRAFT CENTRAL DATA SYSTEM

By T. Baker
G. Cummings
R. South

October 1969

Prepared under Contract No. NAS2-3255 by
APPLIED RESEARCH LABORATORY
SYLVANIA ELECTRONIC SYSTEMS
an Operating Group of Sylvania Electric Products, Inc.
40 Sylvan Road, Waltham, Massachusetts 02154

for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
AMES RESEARCH CENTER
MOFFETT FIELD, CALIFORNIA 94035

NASA CR-73348
Available to the Public

FINAL REPORT

MULTIPAC, A MULTIPLE POOL PROCESSOR AND COMPUTER FOR A SPACECRAFT CENTRAL DATA SYSTEM

By T. Baker
G. Cummings
R. South

October 1969

Prepared under Contract No. NAS2-3255 by
APPLIED RESEARCH LABORATORY
SYLVANIA ELECTRONIC SYSTEMS
An Operating Group of Sylvania Electric Products, Inc.
40 Sylvan Road, Waltham, Massachusetts 02154

for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
AMES RESEARCH CENTER
MOFFETT FIELD, CALIFORNIA 94035

FOREWORD

The study described herein was done at the Applied Research Laboratory of Sylvania Electronic Systems, under NASA Contract NAS2-3255. The work was done under the direction of Mr. Richard O. Fimmel, Systems Engineering Division, NASA-Ames Research Center.

TABLE OF CONTENTS

TABLE OF CONTENTS.-- Continued

TABLE OF CONTENTS.-- Continued

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS.-- continued

LIST OF TABLES

LIST OF ABBREVIATIONS

ACC	Accumulator
A/D	Analog-to-digital
ANAP	Analog amplifier
ANSW	Analog switch
BMIC	Bipolar-to-MOS interface circuit
CCD	Control code
CDS	Central data system
CMD	Command module
CMOS	Complementary metal oxide semiconductor
CNI	Copy next instruction
COMBR	Combinational reliability
CR	Control register
CTR	Counter
DIN	Data input
DOUT	Data output
D/A	Digital-to-analog
DMR	Data memory paging register
DTL	Diode-transistor logic
EX	Execute state of sequence counter
FR	Failure rate
IC	Integrated circuit
IND	Index state of sequence counter
INHR	Inhibit register (signal)
INP/WR	Input or write
INT0	Zero state of interrupt sequence
INT1	One state of interrupt sequence
I/O	Input or output
IR	Instruction register
ISR	Instruction shift register
LIR	Logic instruction register
LSI	Large-scale integration
LSIC	Large-scale integrated circuit
LU	Logic unit

LIST OF ABBREVIATIONS.-- Continued

MA	Memory address (register)
MBIC	MOS-to-bipolar interface circuit
MD	Memory data (register)
MEM	Memory
MNR	Minimum number of modules required
MOS	Metal oxide semiconductor
MR	Reliability of module
MSI	Medium-scale integration
MULTIPAC	Multiple Pooled Processor and Computer
MUX	Multiplexer
N	Number (of modules in system)
NOP	No operation
NS	Number surviving
OPC	Operation code
OUT/RD	Output or read
PC	Program counter
PMR	Program memory paging register
R	Register
REG	Register
R/M	Register or memory
S	Select signal
SC	Shift clock
SQ	Sequence counter
TM	Telemetry module
TTL	Transistor-transistor logic
V_{cc}	Power supply voltage
WS	Word strobe

FINAL REPORT

MULTIPAC, A MULTIPLE POOL PROCESSOR AND COMPUTER FOR A SPACECRAFT CENTRAL DATA SYSTEM

By T. Baker
G. Cummings
R. South

SUMMARY

MULTIPAC is a computer designed especially for use as an "off-the-shelf" central data system for deep space probes. This computer has the unusual characteristic that it may be repaired during flight through the command and telemetry link by reprogramming around the failed unit. This reprogramming is possible through a computer organization that uses pools of identical modules which the program organizes into one or more computers. The interaction of these modules is dynamically controlled by the program and not hardware. In the event of a failure, new programs are entered which reorganize the central data system. The only effect of such reorganization is to reduce the total processing capability aboard the spacecraft. Consequently, some low priority process may have to be eliminated, but data taking and transmission may continue.

As an example of one MULTIPAC configuration, a 16-watt system, including 12,288 words of memory, can act as a sophisticated data management system for a space probe with about 200 science and engineering input lines and 200 output lines. This MULTIPAC system could simultaneously schedule sampling of the experiments, perform needed analog-to-digital conversions, reduce the data using histograms or other data reduction techniques, perform some data processing for the experiments such as digital filtering, and then format the data for transmission by the telemetry subsystem. In addition, the system has all the flexibility of a computer by allowing wide variations in formatting, sampling schedule, etc. These program variations can occur under program control or be completely changed later in the flight from the ground after analysis of the data received.

1.0 INTRODUCTION

This report describes MULTIPAC, a spacecraft central processor, the concept of which was derived from the first year of this study. (The result of the first year study is reported in the final report for that part of the contract.[1]) MULTIPAC has modular organization which permits reprogramming around failed modules. Machine reorganization may be accomplished by program changes to utilize surviving modules optimally, thus affecting a gradual degradation of processing capability as additional modules fail in the course of a long mission. The overall reliability is such that the probability is very high that at least some minimum mode of operating the spacecraft can be sustained throughout very long missions.

The MULTIPAC system is intended to replace the current technique of designing a new central data system for each probe with a standard "off-the-shelf" central data system which is programmed with software to perform as a flexible data management system. Some variation of flight-to-flight requirements are expected to be made up by differences in the number of modules carried and also with the possibility of the addition of one or two special modules.

As an example of one MULTIPAC configuration, a 16-watt system, including 12,288 words of memory, can act as a sophisticated data management system for a space probe with about 200 science and engineering input lines and 200 output lines. This MULTIPAC system could simultaneously schedule sampling of the experiments, perform needed analog-to-digital conversions, reduce the data using histograms or other data reduction techniques, perform some data processing for the experiments such as digital filtering, and then format the data for transmission by the telemetry subsystem. In addition, the system has all the flexibility of a computer by allowing wide variations in formatting, sampling schedule, etc. These program variations can occur under program control or be completely changed later in the flight from the ground after analysis of the data received. In contrast, today's fixed format central data system simply performs a fixed schedule of sampling followed by one of a few fixed formatting routines. Processing of the data is not possible, and the scheduling and the formatting is primarily variable by scaling to the telemetry rate.

The first year of this study, which has been reported earlier, was concerned with overall spacecraft organization and usage of the central data system. The three major recommendations of this phase were that the central data system use stored program computer concepts, data formatting should be very flexible, and data reduction algorithms should be used whenever possible.

Data formatting should be flexible in order to use effectively the telemetry rate when failed experiments are turned off. A format of fixed cyclic sequence of data words is used on present space probes. For this fixed format, the CDS input and output rates are matched to the instrument

sampling rate. The advantages of fixed format are that only a relatively small number of bits (frame sync bits) need to be transmitted to mark the start of the known sequence and, secondly, the same sequence can be used at different bit rates simply by making adjustments in the input sampling rates. The chief disadvantage of such fixed formatting is that, when instruments are turned off, meaningless bits must be inserted into the telemetry data stream in place of those which would normally come from the inoperative instruments to preserve the fixed sequential telemetry pattern. A variable format will eliminate this disadvantage when an instrument is turned off but will pay for this in extra transmitted bits when all instruments are operating. The recommended variable format uses data arranged in source-associated blocks which contain relatively small numbers of bits in a fixed order. Each block carries its own identification bits, which can be distinguished from ordinary data bits. These blocks are then transmitted in a variable sequence.

The first phase of the contract recommended that both the fixed and variable format be available and changes from one to the other be made when experiments are turned off or when telemetering bit rate changes. This ability to carry a number of radically different formats is easily possible if the central data system is a stored program computer.

For some of the experimental data, enough redundancy exists so that data reduction processing can significantly increase the amount of information which may be transmitted at a given telemetry rate. Recommended data reduction techniques are histograms, digital filtering, and spectral analysis. The decision to process the raw data for a particular scientific instrument prior to transmission must be made by the instrument designer or experimenter. Therefore, it is clear that the only reasonable solution is a stored program CDS which could be specifically programmed to each experimenter's requirements.

The central data system is ideally suited to the formation of histograms and the subsequent computation of statistics from these histograms. For cosmic ray and neutron experiments, a histogram of the counts can be accumulated over a large number of spacecraft revolutions. The mean, variance, and modes for each histogram can be computed and transmitted. Alternately, the quantiles of the histograms can be computed and transmitted. Histograms require very little processing time for their implementation, which is desirable in the event of a component failure that would reduce the central data system processing capability.

Another data reduction technique is digital filtering. The availability of a stored program central data system allows consideration of employing digital filtering for replacing analog filters in the instrument electronics. In addition, digital filtering can be employed to reduce signal bandwidth and provide estimates of spectral energy at different frequencies. These filters can be either lowpass, bandpass, or highpass filters.

A third data reduction technique considered in the early study was spectral analysis. Spectral analysis is a mathematical tool for estimating the power spectrum of a time function for a finite length record. There is a basic trade-off when making spectral estimates between the spectral resolution that can be obtained and the variability of the estimate. The finer the resolution of individual spectral lines in a signal, the greater the spread of the confidence range about the estimate. Conversely, reducing the variability of the estimate reduces the resolution of the spectral lines.

Several conclusions can be made about handling data reductions of signals generated by the instruments. In general, it seems better to employ averaging methods (e.g., computational of mean, variance, and spectral distribution) rather than omit data samples when the rate of data collection exceeds the telemetry channel capacity. In this way, the CDS is being used to affect compression, and aliasing errors due to insufficient sampling rates are minimized.

It is also clear that a variety of algorithms could be stored by the central data system so that, when monitoring the data from each instrument, the appropriate algorithm can be selected. This tailoring of the processing of each channel is a distinct advantage possessed by a stored program central data system.

The data formatting and data reduction studies in the first phase of the contract highlighted the need for stored program computer concepts for the design of the Central Data System. A centralized computer for a central data system leads to the problem of how to prevent failures from aborting the entire mission. Reliability becomes even more important when we realize it was recommended in phase one that many additional tasks normally performed in each experiment be taken over by this centralized computer. The solution arrived at was a multiple pool processor and computer (MULTIPAC) made up of a number of modules of a few types tied together by the program. The remainder of this study was devoted to the design of this MULTIPAC system.

The MULTIPAC system, as finally developed, is shown in Figure 1. It's most important module, the Logic Unit, controls the actions of all other module types. Each logic unit, using a few registers and one or two memories, acts as a computer. A typical system will have three logic units and enough registers and memories to act as three simultaneous computers, each performing one-third the overall processing tasks. This typical system will consume only 16 watts and use 173 LSI logic circuits, 768 memory store LSI circuits and six integrated circuits for special purposes (e.g., oscillator). The number of different LSI circuit types is 13 or 17, depending on whether or not a large discretionary wiring LSI type is used on the logic unit.

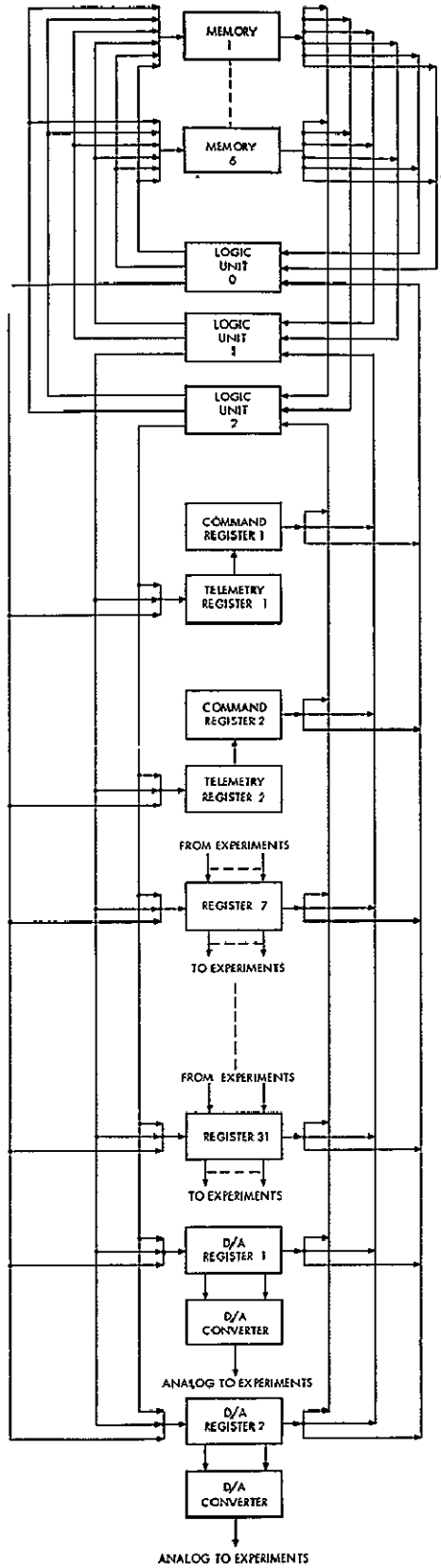


Figure 1. Block Diagram of Typical LSI MULTIPAC System

The registers are dual purpose. They act as index or scratch registers for normal processing and they are the input/output interface to the experiments. Each register contains a separate output buffer which holds output interface information. Thus, the register can be used as a scratch register without disturbing the output interface.

A few of the registers are special. One is used to produce an analog voltage of its digital contents to be used in analog-to-digital conversion for some of the experiments' signals. Another pair is used for the command and telemetry interface. The command register must have the ability to overtake the logic units by command in the event that the system does not respond properly to normal commands due to component failures.

The memories are passive devices which read or write data as commanded by a logic unit. The program counter is contained within the logic unit and instructions are requested from a memory selected as the program memory. Usually, a second memory is selected for data since the instruction rate for a separate data memory is faster than using the program memory for data.

All logic units can address all registers and all memories. In the event that two logic units address the same register or memory, any data transferring to this selected module will be ORed. The hardware normally associated with a multiprocessing system for handling such conflicts was purposely left out to keep the central data system small, light, and low power. Since all three processes are essentially working on three different tasks of the same problem and, therefore, know what the other processes are doing, these conflicts can be kept to an absolute minimum and those conflicts that do exist can be easily programmed around.

Overall, the system described here has a high likelihood of surviving a long mission with small minimal processing capability. A failure of a module will cause slightly degraded processing capability because some extra programming must be done to program around the failed module unless a spare module exists. But even in the case of a failure of a logic unit, the worse that will happen is that the central data system will be able to perform only two-thirds of its processing load. The processing capability of only one logic unit is more than enough to accomplish all the tasks done by the data system of Pioneer VI.

2.0 THE MULTIPAC CONCEPT AND ITS EVOLUTION

The MULTIPAC system was originally proposed as a computer organization which would make the versatility of a programmable central processor available on long space flights without making the spacecraft dependent on the poor reliability of a conventional computer in which any failure normally makes the entire system useless. The solution then envisioned, shown in Figure 2, was a very simple processor organized from modules selected from pools of three basic module types (logic units, memories, and general-purpose registers) which would be assigned to their functional roles by software methods. Should a failure occur, the faulty module could be replaced by assigning another to fulfill its function. Moreover, spares would not have to be assigned as such but could be used insofar as possible to enlarge the initial capacities of the processor. Failures would simply cause a gradual degradation of processing capability so long as sufficient modules remained from which to construct the minimal processor.

As originally envisioned, a processor could be constructed from three logic units, two memories and several registers, plus a multiplexer to provide I/O to the spacecraft experiments and modules to interface with the command and telemetry links. The logic units perform all transfers in the machine as dictated by their individual instruction registers (LIR's). The memories automatically output to their data registers (MD) the contents of the location specified by their address registers (MA) or write in that location any data word transferred into their MD registers.

The data flow, which is programmed, is typically as shown in Figure 3. LIR1, the instruction register of Logic Unit 1, is initially loaded with an instruction causing the contents of a register (R_{pc}), used as the program counter, to be incremented and passed to the address register (MA1) of the memory containing the program.

A second logic unit (LU2) is also initially programmed, causing it to continuously transfer the contents of the program memory data register into the instruction register of a third logic unit (LU3) which actually executes the program. It, in turn, operates on several registers which may be used as accumulators, index registers, counters, and scratch storage, and upon a second memory used for data storage. If the memory address is set in MA2 by the logic unit, one machine cycle later the contents of the location may be read from the memory data register (MD2).

The following table illustrates the overlapping timing with which the program counter is advanced, the instructions delivered into the instruction register and, as an example, how an add from memory into the accumulator (R_{ACC}) is executed. Operation codes used are copy (COP), no operation (NOP), and add (ADD).

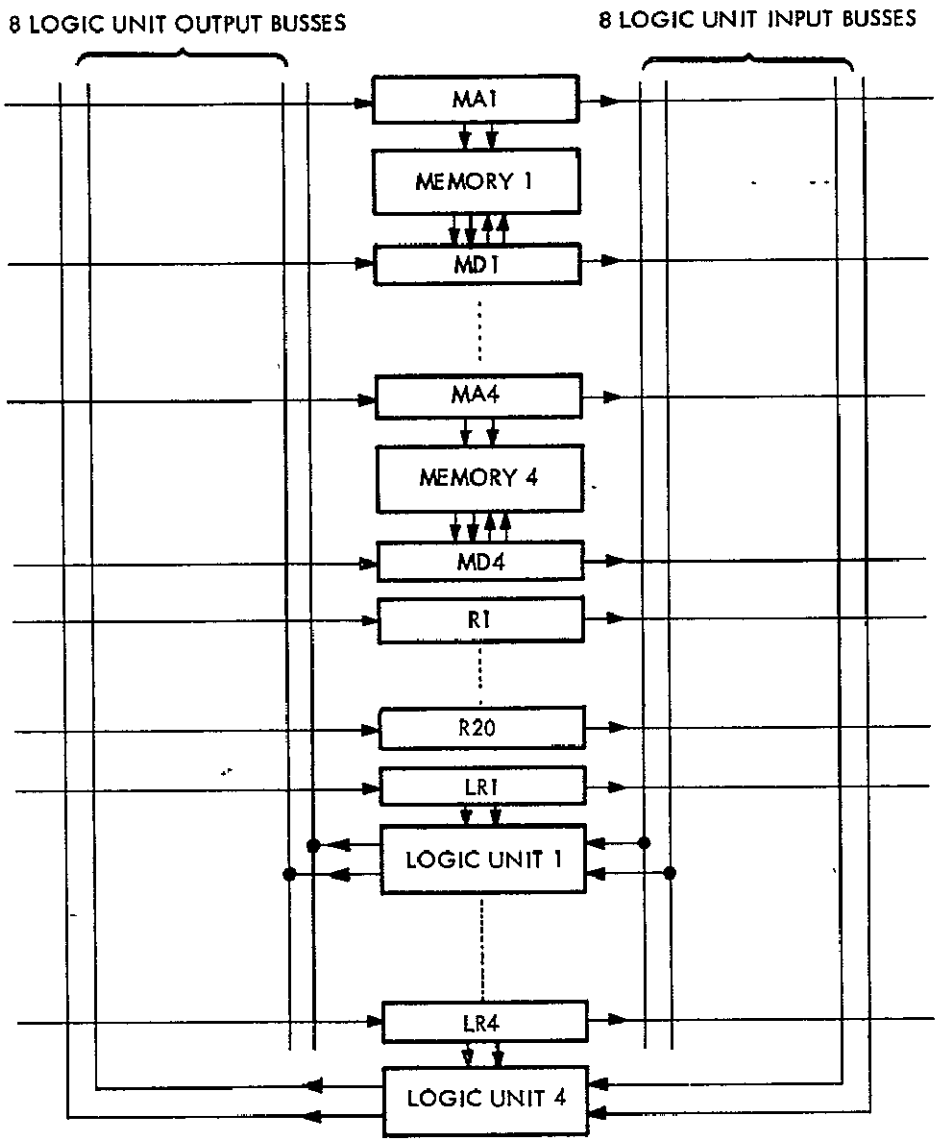


Figure 2. Original MULTIPAC Concept

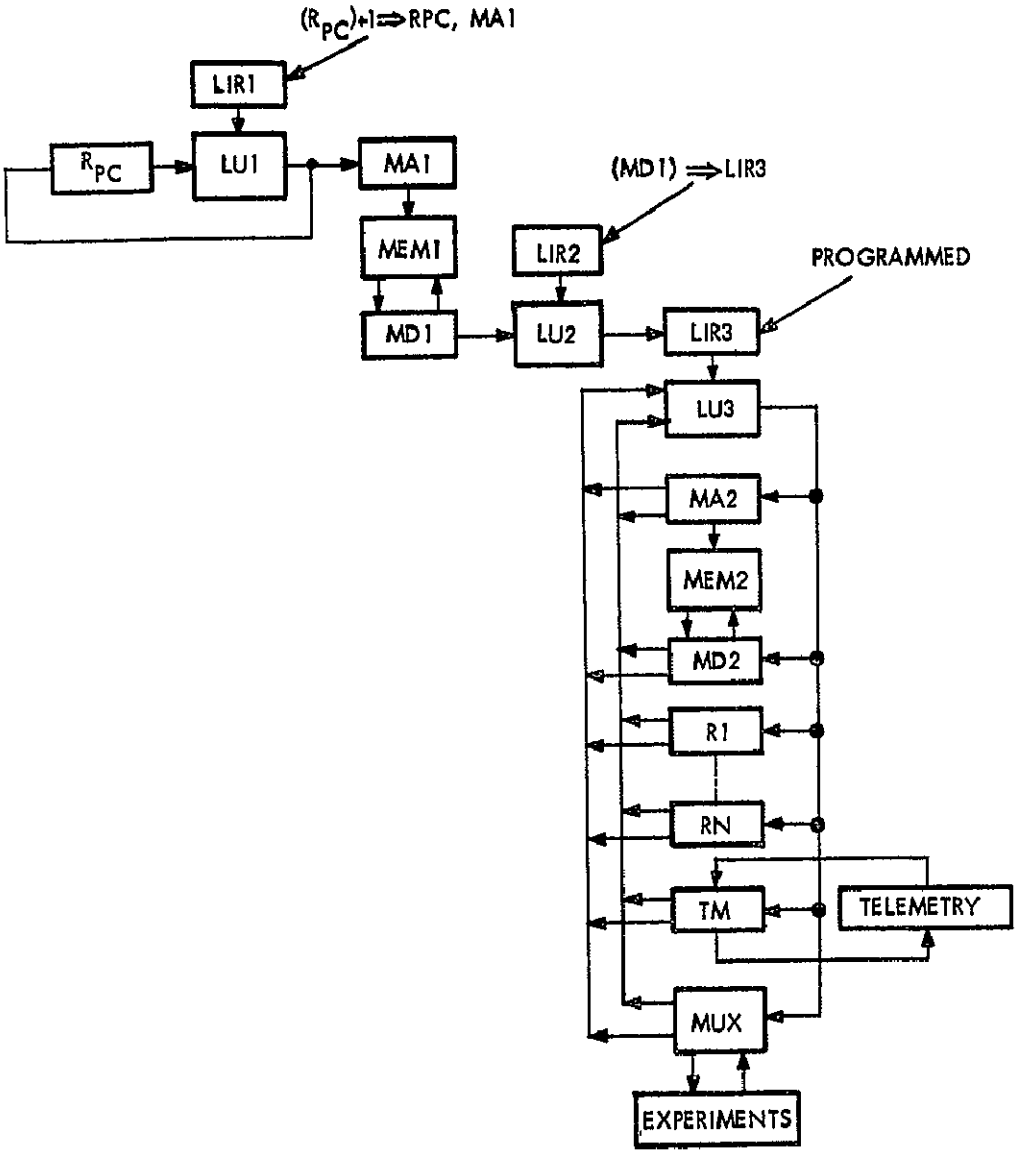


Figure 3. Data Flow in Original MULTIPAC Concept

R_{pc}	MA1	MD1	LIR3	MA2	MD2	R_{ACC}
0	0	---	---	---	---	A
1	1	(0)	---	---	---	A
2	2	(1)	(0)=COP MD1, MA2	---	---	A
---	---	(2)	(1)=NOP ADDR	ADDR	---	A
---	---		(2)=ADD MD2, R_{ACC}	ADDR	(ADDR)	A
						A+(ADDR)

The original machine was to have used a 16-bit word having a 4-bit operation code and three 4-bit addresses. Addresses were 12 bits long with an operation code field containing all zeros, defined as a no operation (NOP) instruction. Addresses were buried in the program stream in what were essentially two-word instructions and prevented from acting as instructions when they reached LIR3 by their NOP coding.

Three factors have proved troublesome in the practical design of the machine. First, there is a great deal of switching interconnecting all the modules in order that they may all be interchangeable. Second, the power limits set upon the design constrain the choice of circuitry to the lowest-powered (and lowest-speed) logic families. Third, the real-time data processing requirement, initially assumed to consist of low-rate data formatting, has grown quite large, enough to exceed the capability of the simple micro-ordered set of trivial modules originally envisioned.

These three factors have influenced the evolution of the design. The choice between a serial or parallel machine was resolved in favor of a serial one, largely in order to minimize the amount of switching logic between modules. Also involved in this decision was the question of speed versus power. Investigation indicated that a parallel machine would have been too large, considering the switching logic, and would exceed the power budget even with very low-powered logic. The serial system was smaller and could stay within the power budget if constructed from very low-powered logic. However, it would be an order of magnitude slower, which would have an adverse effect upon the ability of the machine to handle the processing load. This can be alleviated to some extent by the use of a small percentage of higher power, faster logic in the critical data paths.

The power budget was also responsible for the decision to reduce the machine word size to 12 bits. The three-address instructions were eliminated, which not only disposed of one 4-bit address field, but also prohibited instructions designating two locations in which the result should be stored. This permitted the simplification of the switching logic to include only one output data bus instead of two.

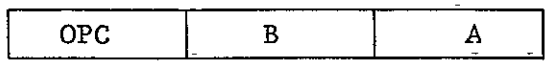
The reduction in gating resulting from a serial transfer organization was further reduced by eliminating the normal transfer control lines to each of the registers. Instead, the number of transfer pulses was set at 15; one for word strobe (beginning of word), two for transfer control, and 12 shift pulses for the 12 data bits. The two transfer control bits inform the register what to do for the 12 data shift pulses (e.g., read word in, read word out) and replace the normal gated control lines. The extra logic due to encoding and decoding the 2-bit transfer code is more than offset by the savings in gates needed to switch the control signals.

Considering the speed that could be attained with such a serial machine (estimated at about 15-microsecond instruction times), it became necessary to consider a multiprocessing system having two or three independent processors in order to fulfill the real-time requirements. This, in turn, increased the number of modules required and the size of the switching matrix. At this point, it became necessary to depart from the generality of the Figure 2 arrangement, which had standard logic unit modules doing such simple tasks as incrementing the program counter and transferring the output of the program memory into the instruction registers. The logic units used for these simple tasks were eliminated. Now, self-incrementing logic is built into the memory address register and the program memory selection switch is built into the logic unit. The number of logic units was reduced by two thirds. Since logic units are no longer program addressable devices, a secondary bus structure was created for the transfer of instructions. In essence, some of the simplicity and generality of the original concept had to be specialized to meet the demands of speed and efficiency.

MULTIPAC evolved from a processor in which each microinstruction was dealt with separately and was independent, so far as the hardware was concerned, from those preceding it and following it. Thus, its individual treatment by the instruction register had to be coded into it, and sequences of microinstructions were solely a matter of programming. As a more specialized system evolved, however, sequences of instructions had to be anticipated in the jump and interrupt hardware. This specialization was extended to realize further efficiencies by recognizing other similar sequences of microinstructions, or macroinstructions, in the hardware. These macroinstructions would be multiple word instructions containing memory addresses. The sequencing hardware need only recognize these addresses and prevent their being treated as instructions.

It appeared useful to be able to use the same memory for instructions and data, thus permitting more flexibility in the modes of MULTIPAC operation, including a single memory mode with a higher probability of survival than any previous workable configuration. This was accomplished through macroinstructions using a program counter (PC) separate from the memory address register. It increments its own contents, delivering the result back to itself and at the same time transferring it to the memory address register. Jump instructions require a control signal to alter the PC as well as the MA. Other addressing of the MA overrides the input from the PC to the MA but does not affect the actual PC contents.

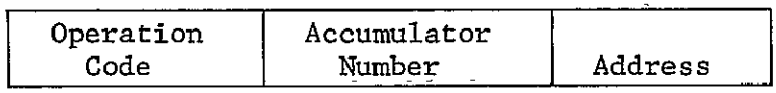
The address fields of the instruction format were increased from four to six bits in order to address all devices directly. The previous machine required 40 addresses. Six-bit fields would provide an expansion capability of 24 addresses, a feature which is highly desirable in the system. Additionally, the use of macroinstructions suggested above would require at least five or six bits in the operation code field. It initially appeared that if three fields were retained in the instruction, the machine word would have to be lengthened to 18 bits, as follows:



6 bits 6 bits 6 bits

An earlier design used 4-bit register address fields with 3-bit base registers to give an effective 7-bit register address. To change from the 16 addresses each instruction could address, the base register would have to be changed by an extra instruction. It was quickly discovered that (1) many of the 16 addresses (particularly memory) had to be independent of base register setting to prevent excessive base register settings, and (2) the few addresses left per base register significantly inhibited efficient programming, especially in the event of a register failure.

The machine word could be reduced to 12 bits, however, by reducing the number of address fields to one. The reason for the two-address instruction, if traced back, lies largely with the earlier use of one logic unit to transfer instructions into another logic unit. Normal data operations could function with only one address field if the logic unit contained two accumulators and a 1-bit field in the instruction referencing one or the other. Indexed or indirect addressing operations require a temporary storage and addition facility in the logic unit whose use will not destroy the contents of the regular accumulator, i.e., a second accumulator. This gave an instruction format of:



5 bits 1 bit 6 bits

In the course of designing the I/O devices, it was found that the multiplexer had switching problems very similar to those of the logic units. At least 200 I/O channels had to be provided, which called for something like four modules, each containing addressable 64-way switching of in, out, and control signals. This problem was solved by combining the multiplexer with the general-purpose registers and providing special instructions to use each bit of each register as an I/O channel. The number

of channels required was essentially divided by 12, the number of bits in each register, and the multiplex switching was moved back into the main address switching.

Another step taken to increase the efficiency of the processing routines was the improvement of the A/D conversion method. The original concept called for analog-to-pulse-width converters at the experiments, the duration of whose output levels would be counted by the processor. This consumed too much processing time, since the processor would have had to devote itself to each conversion for about 50 milliseconds in a tight loop to achieve the required 8-bit accuracy. To solve the problem, a new module was created, a register which has a D/A ladder network on its outputs. This provides an analog reference signal to the experiments, and each of the latter now must have an analog comparator which returns a signal level to the processor I/O indicating whether their output signal is greater or less than the reference signal supplied. This makes possible the successive approximation method of conversion whose algorithm runs much faster, i.e., about 1.4 milliseconds for the maximum length (8-bit) conversion.

The instruction set was then enlarged from simple two-argument logical and arithmetic instructions and test commands to include the bit manipulating and I/O instructions necessary to make the I/O registers and the A/D algorithm operate efficiently.

3.0 SYSTEM OPERATION

MULTIPAC is expandable and is comprised of seven module types as follows (see Figure 1 in Introduction):

Module	Min. No. Required	Number in Typical System	Number in Fully Expanded System
Logic Unit	1	3	5
Memory Unit	1	6	15
I/O Register	1	25	57
D/A Register	1	2	2
Command Unit	1	2	2
Telemetry Unit	1	2	2
Timing Generator	1	1	

A processor is formed by software assignment of one logic unit, one or two memories, and several registers to operate in conjunction with one another. The I/O Registers serve the purpose of index or temporary storage registers and also provide I/O connections to the system. The most efficient use of memory units requires two per processor in order that program storage and data storage can be separate. The processor can also operate from a single memory unit but at a reduced computation rate. Communication with the command receiver and the telemetry transmitter are provided by the Command and Telemetry Units respectively, which are essentially specialized registers. Another specialized register is the D/A Register, which has a D/A ladder connected to its outputs so as to provide a reference signal for the successive approximation method of A/D conversion.

The MULTIPAC System is not limited to the module types listed above, but these are sufficient for our "typical" mission. Possible omissions are a magnetic tape (or other mass memory) interface, a real-time counter, a sample rate counter, and a Television imaging system. These are discussed briefly in paragraphs 3.4.8, 3.4.9 and 3.4.10 below, and logic diagrams for a real-time counter and a sample rate counter are described in paragraphs 5.11 and 5.12.

In the discussion which follows, the system is assumed to be configured with the quantity of modules listed in the table above under the heading "Number in Typical System."

3.1 Data Flow

Data flow takes place only under the control of one of the three logic units. Each of these communicate directly with six memories, 25 general purpose registers which also serve as I/O interfaces, two D/A registers, two command registers, and two telemetry registers.

Each memory is controlled by the logic unit addressing it. The logic unit can direct the memory to read and send the contents of a specified memory location to itself as either instructions or data, or to write the data which it supplies to the memory.

Each of the 25 general purpose registers has connections for 12 digital inputs and 12 digital outputs which may be accessed by I/O instructions. Thus, they provide 324 inputs (counting the D/A registers) and 300 outputs to the rest of the spacecraft and the instruments.

The D/A registers are very similar to the general purpose registers, having the same number of input channels, but the output channels are not present. Instead, a D/A ladder network is connected to provide an analog signal proportional to the arithmetic value of the register contents. This analog signal is used by the experiments as a reference voltage in the successive approximation A/D conversion process.

The telemetry and command registers share a common address. Instructions operating on such an address will connect one of the command registers to its input bus and/or one of the telemetry registers to its output bus.

The command module serves as the link between the command decoder and receiver and the MULTIPAC system. The module has three purposes: To transfer normal commands (e.g., turn on or off experiments, change mode), to allow special override commands to diagnose and reload new programs from the ground through the command link, and to allow loading of programs while on the ground before launch. The latter two use the ability of this module to have the instruction words it receives executed while inhibiting the normal program stream.

3.2 Transfer Timing

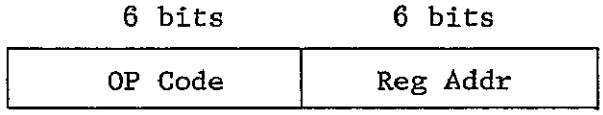
All data transfers between modules are serial. Synchronous machine timing is provided by two clock signals, the shift clock (SC) and the word strobe (WS). These are generated in the Timing Generator and distributed to all modules by triplicated signals driving majority voting gates at each module interface. Timing consists of 14 SC pulses followed by one WS pulse at equally spaced intervals of approximately 1 microsecond. The machine cycle is therefore about 15 microseconds.

The actual 12-bit data transfer is preceded by the transfer of a 2-bit control code on the same line. It is by the transfer of this code that the logic unit controls the operations of the other modules with which it communicates.

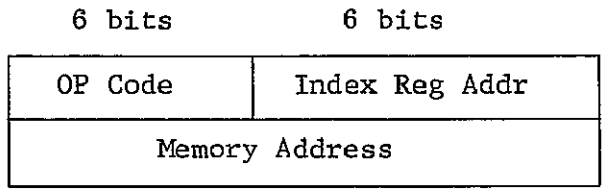
3.3 Word Format

MULTIPAC uses a 12-bit word. The instruction format is as follows:

Single Word Instruction:



Double Word Instruction:



Data words are 12 bits in length and use two's complement arithmetic.

3.4 Module Types

3.4.1 The Logic Unit.-- This module executes the program it receives from the memory unit which it selects as its program source. Figure 4 is a block diagram of the Logic Unit. It is connected to all other modules in the system and controls those which it addresses. In general, a module is addressed by only one logic unit, the one to whose process it is assigned. (Use of a module by more than one processor for purposes of intercommunication must be coordinated between the two programs concerned.) The Logic Unit selects one memory as its source of program and another (although it may also be the same one) as its source of data locations by means of an instruction which loads two 4-bit base, or paging, registers.

The Logic Unit also addresses 64 register locations by the contents of the instruction R field, or six lowest order bits. The first seven such locations are specifically assigned as follows:

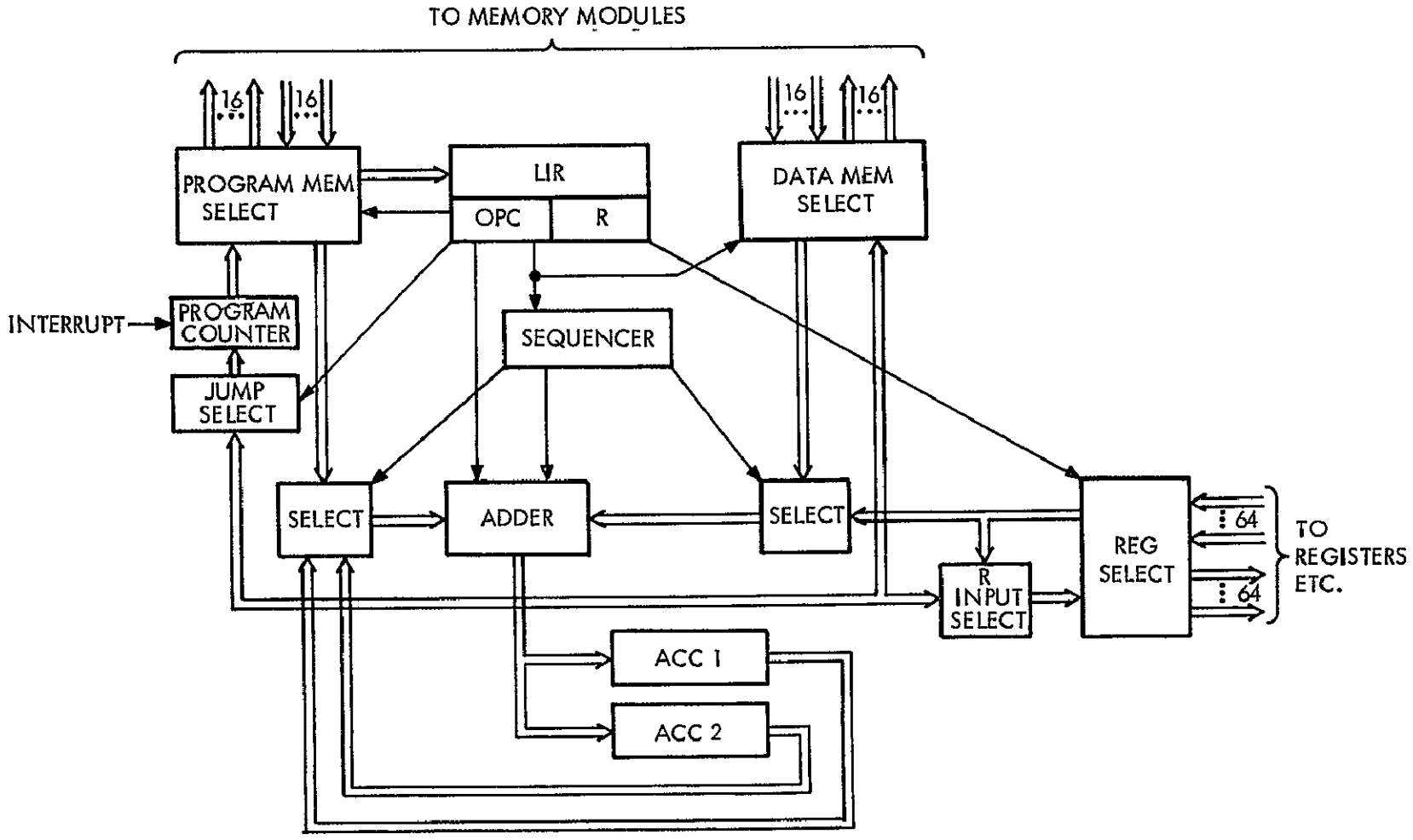


Figure 4. Logic Unit Block Diagram

Address	Register
0	Dummy register: Contents = 0
1	Accumulator 1
2	Accumulator 2
3	Input: Command Unit 1
3	Output: Telemetry Unit 1
4	Input: Command Unit 2
4	Output: Telemetry Unit 2
5	D/A Register 1
6	D/A Register 2

The remaining addresses in the first addressing section are nine. Address switching may optionally be included to expand the number of addresses in blocks of 16 to the maximum of 64. The unallocated register locations, up to 57, will normally be assigned to I/O registers, which makes the permissible I/O interface as large as 57 x 12 = 684 channels each way. These registers also serve the functions of index registers and provide scratch storage for the processor.

Arithmetic and logical operations are performed with the contents of either the registers or the data memory by means of a serial adder and two internal accumulators. Section 7 describes instructions executable by a logic unit. All instructions which access memory are two-word instructions and require two memory cycles for their execution, assuming a data memory unit separate from that in which the program is stored. If only one memory unit is in use, i.e., if the contents of the program paging register and the data paging register are the same, the instruction cycle is automatically extended one cycle. All memory accesses are in practice indexed. Non-indexed instructions reference index register zero and the contents of the dummy register R0 are hardwired to present the number ZERO.

Each logic unit has interrupt capability which may be enabled by an EINT instruction which sensitizes that particular logic unit to the interrupts. Upon responding to an interrupt, the logic unit breaks off the program stream, and the interrupt hardware forces the program memory address to zero and executes the instruction in that location before modifying the program counter. The instruction stored in location zero (STPC, see Section 7) will store the program counter in a register. After executing this STPC instruction, the program counter is set to ONE and execution of instructions proceeds from there.

Since a second interrupt during the interrupt subroutine would destroy the return address, the interrupt enable flip-flop is cleared at the start of each interrupt and must be enabled before returning to the main program.

3.4.2 The I/O Register.-- Figure 5 is a block diagram of the I/O Register. It consists of a shift register for shifting serial data to and from the logic units, an output buffer register for holding output interface information, gating to enter input interface information into the register in parallel, and control section which decodes register control codes from the logic units. The 2-bit control codes cause one of the following four actions:

Control	Action
00	Do nothing.
01	Read input channels, then shift register.
10	Shift register.
11	Shift register, then load output buffer.

For the 10 code, the register is simply shifted which causes data to be serially read into the register from the logic unit and into the logic unit from the register. The 01 code reads the input interface data into the register and then by shifting the register, sends the data to the logic unit. The 11 code shifts serial data from the logic unit to the register and then transfers the data in parallel to the output buffer. The 00 code does nothing. In the case of inputting or outputting, it also generates clock pulses to the receiving or outputting I/O devices to acknowledge the transfer of I/O data to or from them. In the case of serial transfers, for example, these serve as-shift pulses shifting data into or out of the I/O device.

It should be pointed out that the Register Control Section is also intended to serve a similar function in the Memory Unit. What differences exist in the two functions are accommodated by hard-wiring two connections, REG and MEM, to +Vcc and ground respectively when the circuit is used in the I/O Register module. The inverse connections are made when using the circuit in a memory unit.

3.4.3 Memory Unit.-- The Memory Unit incorporates the I/O Register module, with minor alterations to its timing through hard-wired connections, together with additional LSI circuits for address decoding, interface circuits and complementary MOS memory storage cells. Figure 6 is a

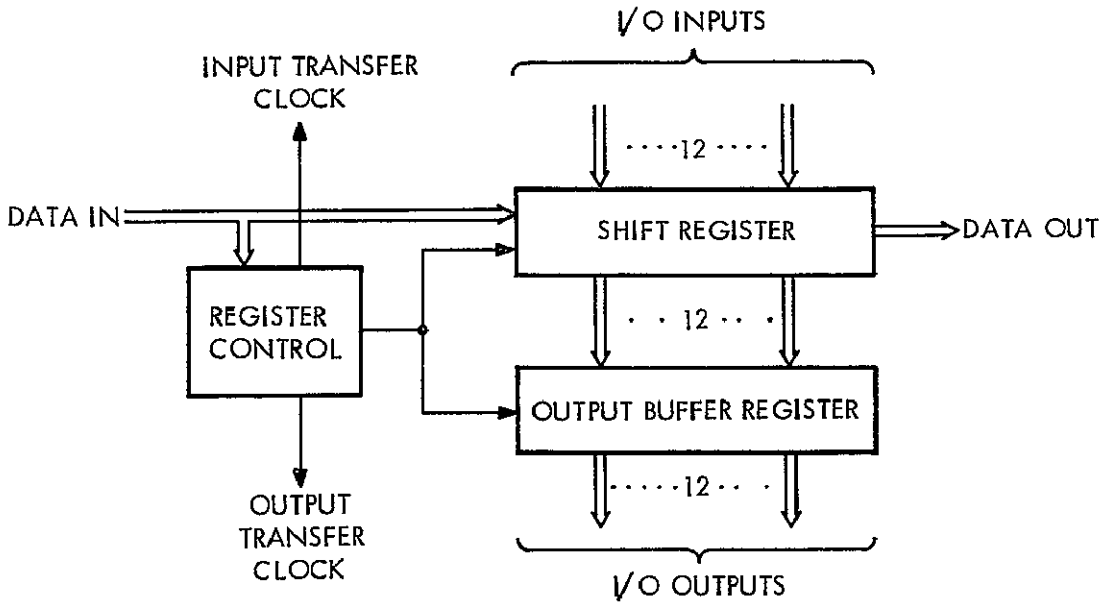


Figure 5. I/O Register Block Diagram

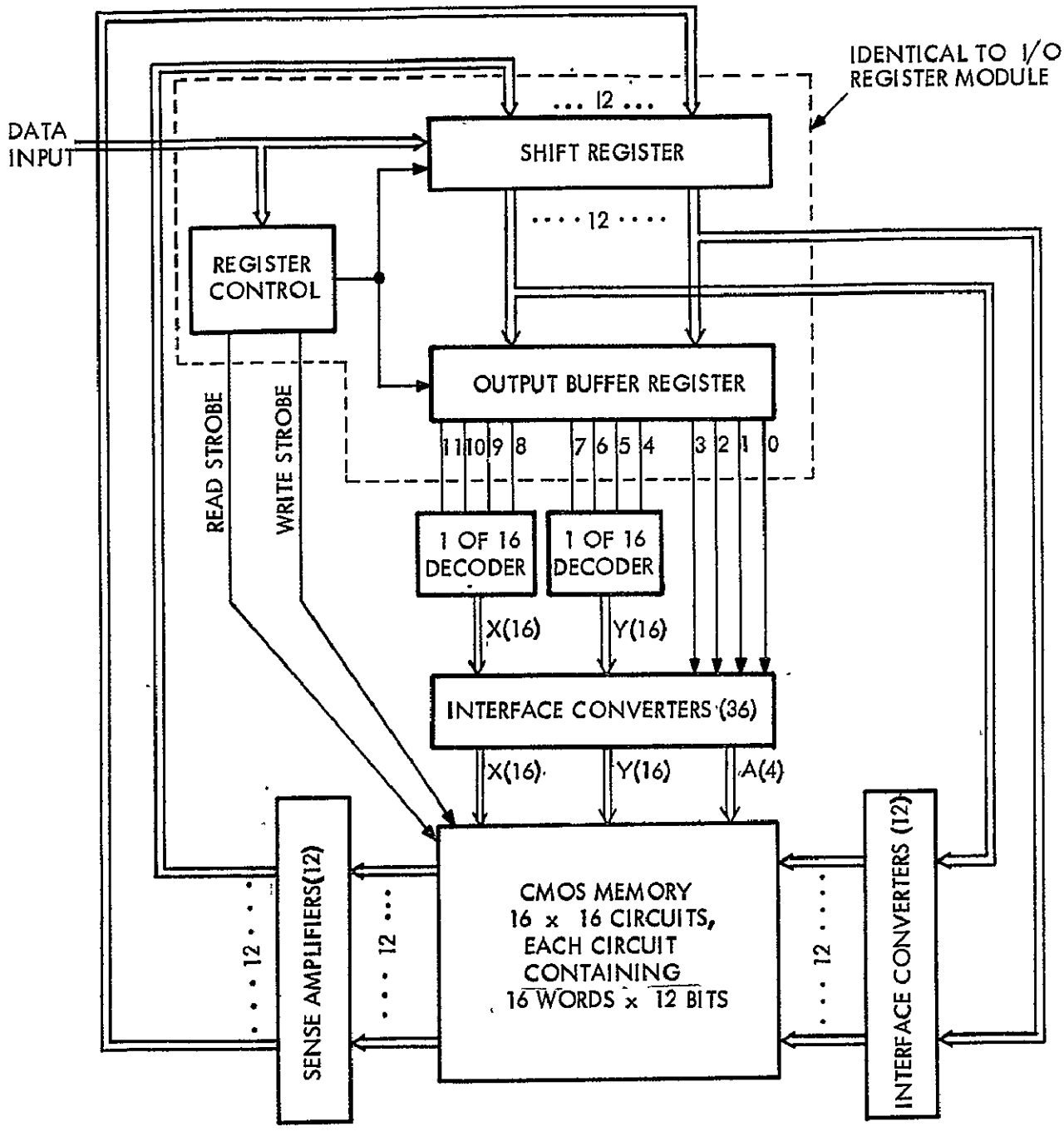


Figure 6. Memory Unit Block Diagram

block diagram of a 4096-word unit. The X and Y lines select one circuit of 16 words and the A lines are decoded at the circuit to select one word. The 2048-word memory as used in the typical system would have only 8 X lines and 28 interface converters.

Complementary MOS was chosen for the basic storage element as the only element with low enough standby power and operating power to allow large amounts of memory for the MULTIPAC system. This element has a standby level of less than 100 nanowatts per bit and can be driven with relatively low power drivers (order of 10 milliwatts). It is commercially available in 16-bit memory chips and NASA ERC has a 256-bit element under development.[a]

3.4.4 D/A Register.-- The D/A Register is similar to the I/O Register, lacking only its output buffer register and replacing this with an 8-bit D/A ladder network. Figure 7 is a block diagram of this module. It supplies a reference signal to all peripheral devices requiring A/D signal conversion. Each such device has its own comparator to compare this reference signal with the analog signal to be converted and returns the resultant bilevel signal into one of the I/O channels which indicates whether the reference signal is greater than or less than the analog signal to be converted. The processor then tests this channel as it performs a programmed A/D conversion by the successive approximation method. Since the D/A ladders are connected directly to the shift register stages (as opposed to the buffer register), the MSKR instruction and other register instructions can be used for this conversion routine.

3.4.5 Command Unit.-- The Command Unit provides an interface with the command receiver to receive normal operational commands for the spacecraft and also to take over control of the processor(s) for reprogramming. Figure 8 is a block diagram of the unit.

The command receiver must assemble a digital word of data, and load this word into one of the two CMD registers in the CDS. Commands received on the up-link will contain a special command address of four bits. These will directly address the instruction register and both accumulators in each of five possible logic units in such a manner that they can override their normal functioning and force data into them. The sixteenth address is used for normal command transfers.

3.4.5.1 Normal commands:-- Normal commands are handled by the use of the sixteenth command address. Receipt of this address causes a program flag to be set through one of the I/O channels and the command word itself remains in the command unit shift register until read out by the program.

---

[a] See Reference 3 at the rear of this report.

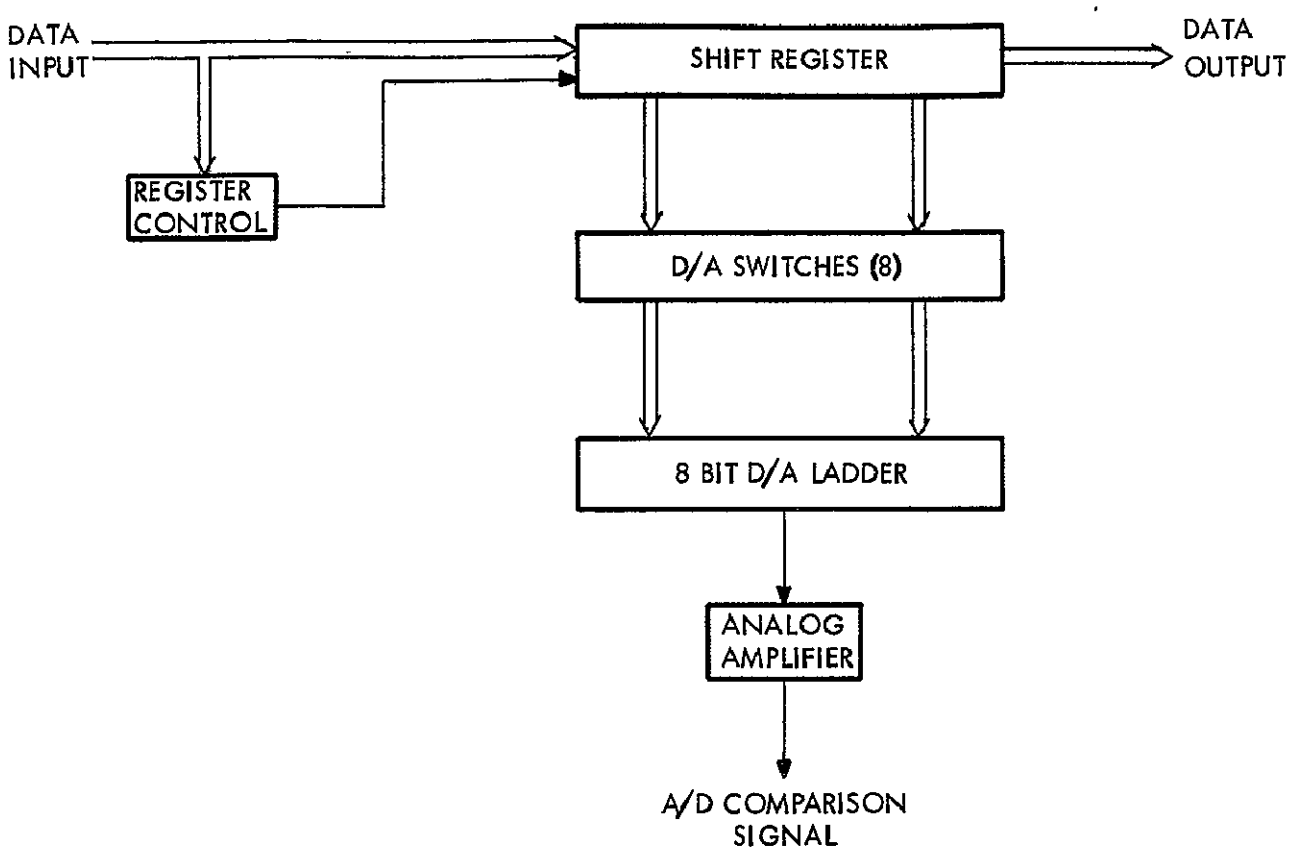


Figure 7. D/A Register Block Diagram

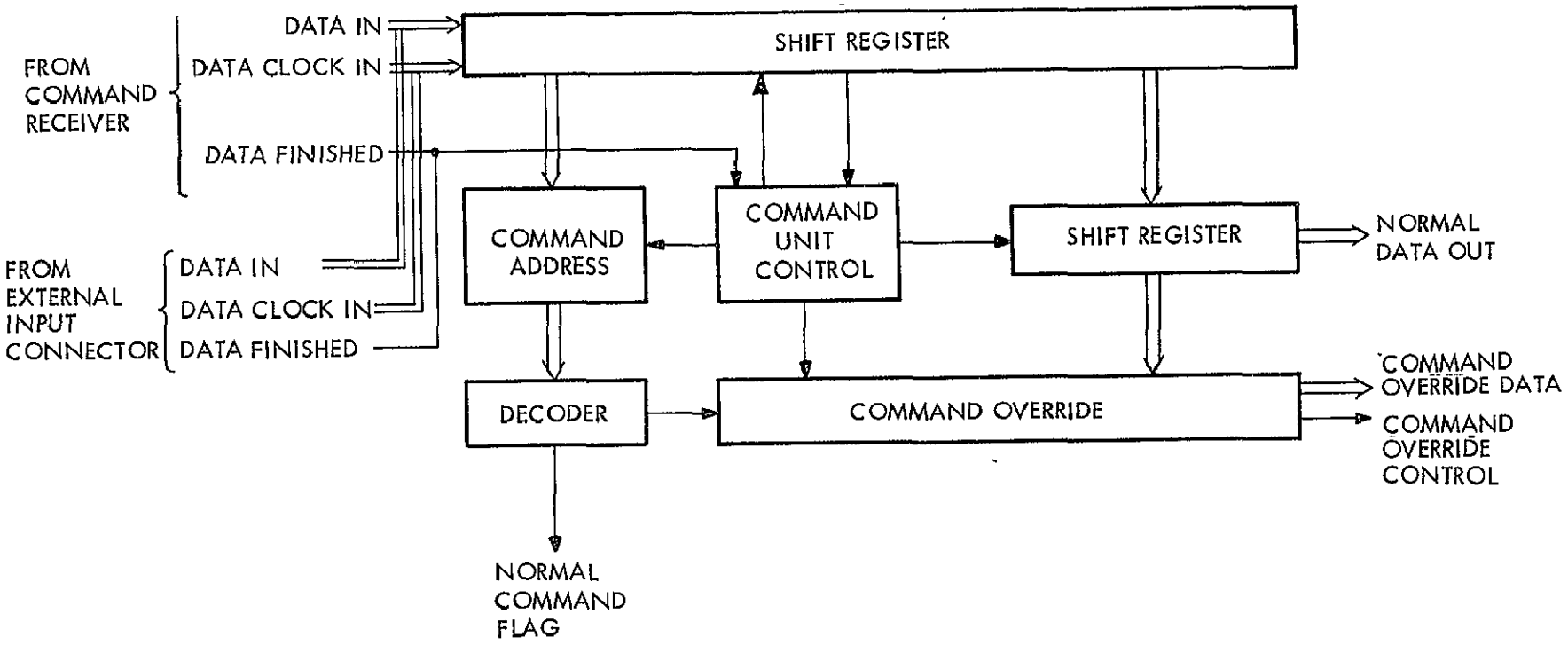


Figure 8. Command Unit Block Diagram

3.4.5.2 Command override:-- The command module inputs to the program switch exercise hard-wired priority over the source dictated by the command address bits. This overriding control is used to take over the MULTIPAC system to recover from circuit failure by reprogramming.

The command override function replaces the normal program source of the logic unit and causes instructions thus inserted to be executed in lieu of the next program step. The first step in reprogramming is to introduce an SPMP (Set Program Memory Page - see Section 7) instruction into each logic unit, setting the program page register to ZERO. This position is unused and is hard-wired to a data level of ZERO. All logic units would therefore copy instructions which are all ZEROS. This is interpreted as a series of no operation (NOP) instructions and the logic unit is effectively disabled. Since only the logic unit itself can address its own program switch, there is no danger of another processor which is still active interfering and restarting it. During the relatively long period while each instruction of the reprogramming bootstrap loader is being received on the command link, the logic units are disabled, but when an instruction has been assembled in the command receiver and transferred to the MULTIPAC Command Register, it is inserted in the stream of NOP's at the normal instruction rate.

A bootstrap loader is then written into one of the memory units through one of the logic units. This memory unit is selected to be the data memory of the logic unit, and the memory address and data to be stored there are loaded directly by the command override logic into ACC2 and ACC1 of the logic unit. An instruction is then loaded by the override logic directly into the instruction register to store in data memory the contents of ACC2 indexed by ACC1. The address of the store instruction will be the next word seen in the program, but since this is all zeros except for instructions inserted by the command override, the address seen will be ZERO. Thus, the data word contained in ACC1 will be loaded into the location specified by the index in ACC2. Once a bootstrap loader has been stored in a memory unit, the program paging register can be switched to that unit by the command override and the remainder of the new program loaded by means of normal command transfers.

3.4.5.3 Loading from the ground:-- Programs can be loaded on the ground before launch through the input connector shown on Figure 8. This connector is wired in parallel with the signals from the command receiver. When the receiver is off, the programs can be loaded in the same manner as the command override, except at a much higher rate since there is no command link limitation. Of course, once the program is loaded, the power to the memory must remain on to retain the information.

3.4.6 Telemetry Unit.-- The Telemetry Unit interfaces with the modulator of the telemetry transmitter which is used to transmit the spacecraft data to the ground station.

The block diagram of this unit is Figure 9. It is similar to an I/O Register except for the Telemetry Buffer Register and associated logic. The buffer register shifts, including a 1-bit high-order extension of it, on the telemetry clock pulses. Since the 1-bit extension is preset to a ONE but shifting fills from the left with ZEROS, when the ONE reaches the next-to-low-order stage, the contents of the register will be either 0002_8 or 0003_8, depending on the last bit of telemetry data. At this point one more shift would bring the preset ONE to the telemetry interface. Instead, however, the control causes the next telemetry clock pulse to load a new telemetry data word in parallel from the shift register and to preset the extension bit again. A flag to the processor is also set to advise it that the next telemetry word should be transferred into the shifting register. The frequency with which the processor must sample the flag is one-twelfth the telemetry bit rate.

3.4.7 Timing Generator.-- The Timing Generator, diagrammed in Figure 10, provides both the shifting clock (SC) pulses and the word strobe (WS) pulses to all other modules. Each of these signals is supplied in triplicate throughout the system and is decoded by majority voting gates at each module interface. The Timing Generator contains two sources of 1-MHz* square waves selectable by the Command Decoder plus three identical counters which operate in synchronism, routing 14 of the clock pulses onto the SC line, then diverting one to the WS line and resetting. This resetting, which maintains synchronism, is accomplished through majority voting gates also. Thus, the clock distribution system can absorb the malfunction of any one of these counters, or clock drivers, or the loss of any one clock signal up to the individual module interfaces.

3.4.8 Real-Time Counter.-- Many missions will require a real-time counter in order to label experimental data with time of occurrence. This will be particularly true if data is stored or data reduction techniques performed before transmission to the earth. The real-time counter designed for MULTIPAC is expandable in increments of 12 bits. Figure 11 shows a block diagram of a 36-bit real-time counter. Thirty-six bits will cover a time span of about two weeks with a precision of 15 microseconds. The real-time counter can be implemented with only one new LSI chip type shown on the block diagram of Figure 12 as Increment And Control. This circuit allows the top shift register to increment once every word-time and the bottom two shift registers to increment on the word-time following an overflow of the shift register immediately above. In addition, this circuit will select one of the three shift registers as an output on receiving an input command (INP instruction) from a logic unit. These input commands select each shift register cyclically. If an INP instruct selects the top shift register after the next application of the INP instruction to the same register control, the middle shift register will be selected and then the bottom shift register will be selected. An OUT instruction will cause the next INP instruction to select the top shift register. The select circuitry shown as a separate block is actually packaged in the Increment And Control LSIC.

---

* The actual clock rate should be 983.04 kHz to obtain a 2^{16}-Hz word rate if a real-time counter module is present.

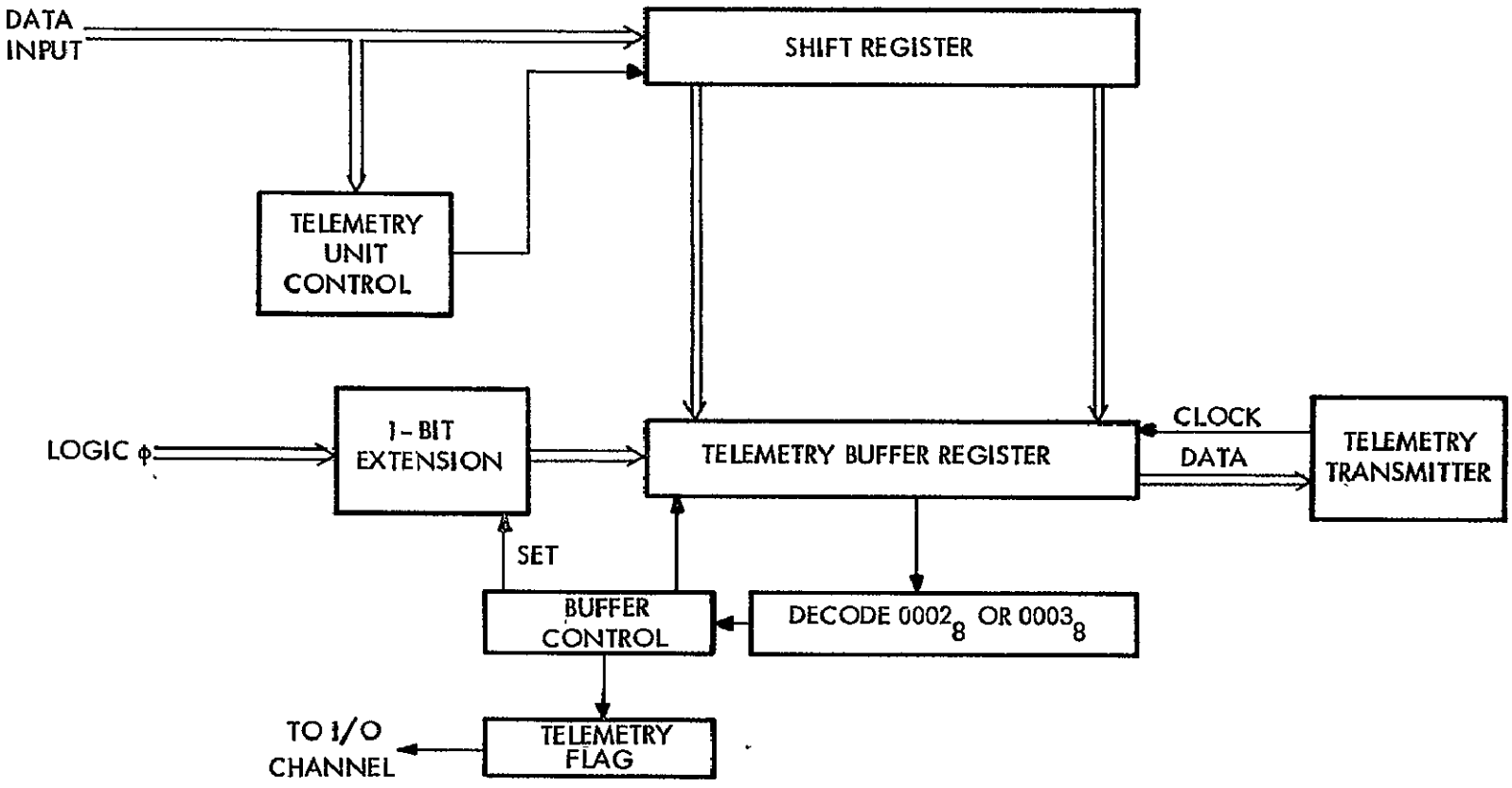


Figure 9. Telemetry Unit Block Diagram

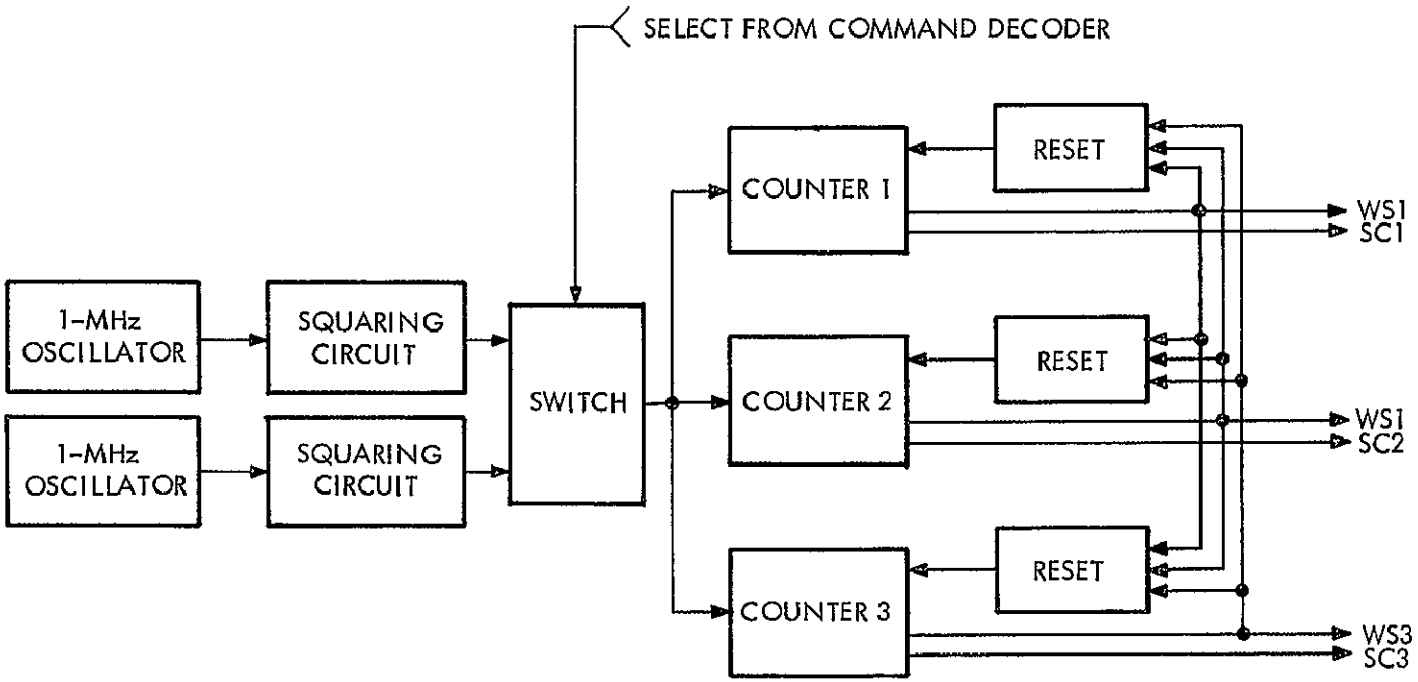


Figure 10. Timing Generator Block Diagram

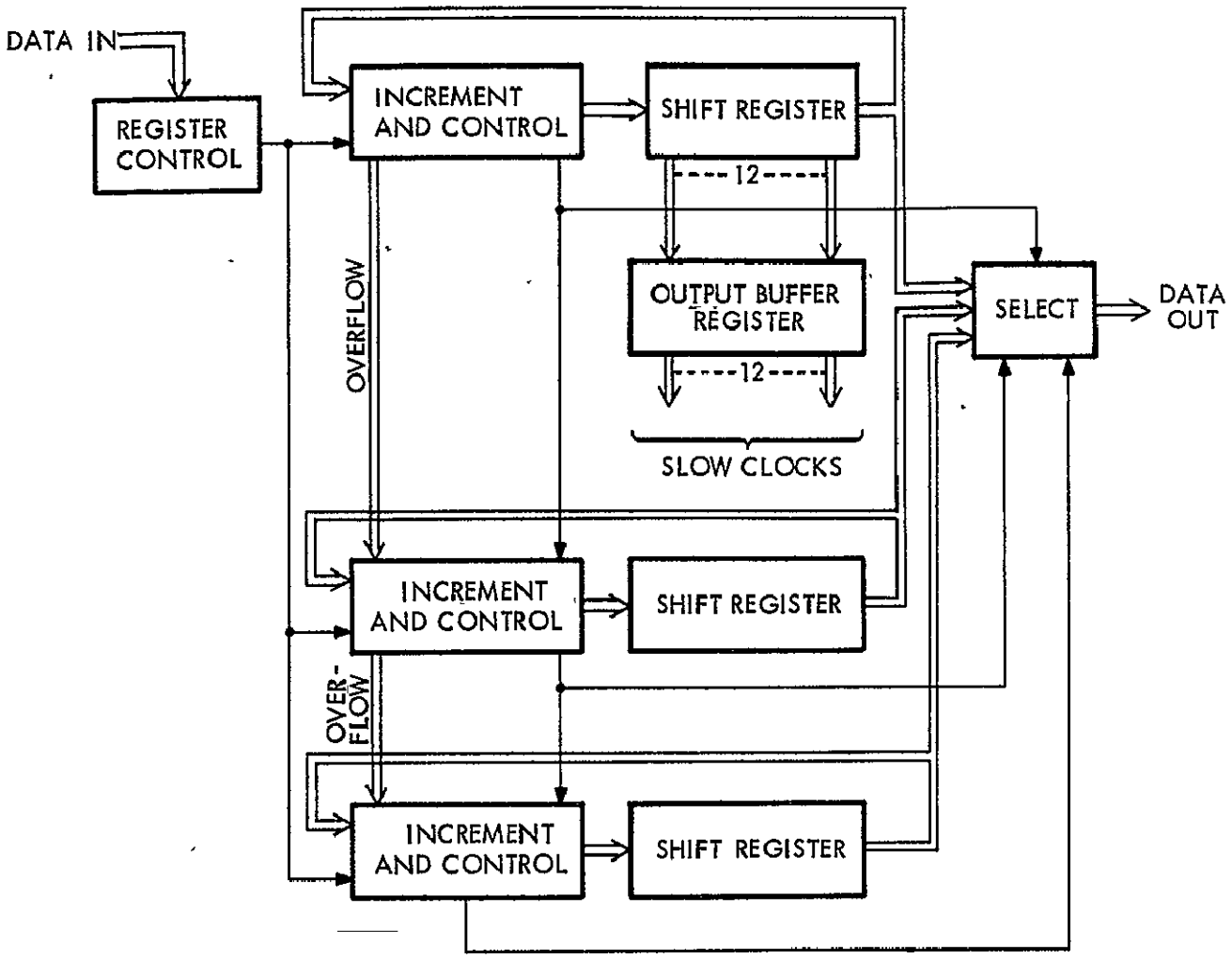


Figure 11. 36-Bit Real-Time Counter

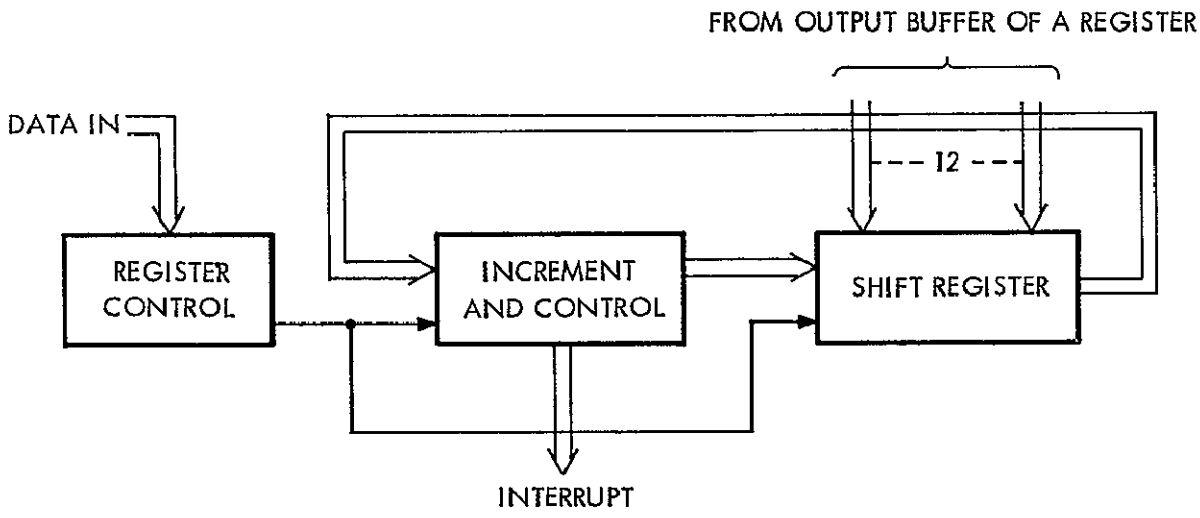


Figure 12. Sample Rate Counter

Clocks slower than the 65,536-Hz MULTIPAC word rate can be generated easily by the addition of one or more output buffer LSI circuits to the real-time counter. One such buffer is shown in Figure 11. This buffer is loaded every word rate. Each stage, therefore, represents a clock frequency of 2^n Hz where n varies from 4 to 15 (16 Hz to 32,768 Hz). Slower rates can be obtained by adding another buffer to the middle shift register. These slow outputs will be used by the telemetry receiver and can also be used by the experiment.

3.4.9 Sample Rate Counter.-- Many missions could use one or more sample rate counters to control sampling rates of experiments. One very obvious use of such a counter is to time out a sector from the sun pulse generator. A 12-bit version of the sample rate counter is shown in Figure 12. The only new LSI circuit is the same as that required for a real-time counter (see paragraph 3.4.8), which has been designed for either use, depending on external connections. For count rates slower than that obtained with 12 bits (16 Hz), these may be expanded in a manner similar to the real-time counter or may be triggered from an overflow of a real-time counter. The sample rate of this counter is stored in an output buffer of another I/O register. This count need only be read into this output buffer with an OUT instruction once. The shift register is incremented continuously, and everytime the shift register overflows, the 12 bits of this output buffer are jammed into the shift register and counting commences from this number. When the count overflows, an interrupt is also generated to be tied into the interrupt input of the logic unit. This interrupt flip-flop will be cleared when an OUT instruction occurs at this register. This register may also be resynced with an INP instruction which reloads the register with the starting number.

3.4.10 Magnetic tape unit (not implemented).-- It is likely that future spacecraft missions will have a magnetic tape unit aboard as a mass memory. In the past, these have been operated to look very similar to the telemetry interface. The data is transferred to the unit (and also stored) as low-speed serial data. If this technique continues, then a module with characteristics similar to those of the Command/Telemetry Unit (considered as a whole) could be designed, or the Command/Telemetry Unit changed (if needed) to accomplish both command/telemetry and magnetic tape interface.

If, on the other hand, a magnetic tape unit is designed to take advantage of a stored program central data system, then the interface will depend on the characteristics desired. The simplest interface, in terms of hardware, is to use the standard I/O interface of MULTIPAC. To handle reasonable transfer rates, this would require transferring 12 data bits in parallel (i.e., using all output buffer bits of one register) and using additional input and output channels from another register for control channels.

The most likely magnetic tape interface would have a high-speed serial data transfer with control of the tape to go forward or back at one speed. However, regardless of what the interface looks like, a special additional module could be designed which could connect into the MULTIPAC

system without change of the system. This module would use the register control circuit used by the I/O Register module to interface to the register bus. The three different control codes (shift, input, and output) could be used to distinguish between writing, reading, and tape control functions. For the tape control words, the 12 bits become commands to change modes to reading, or writing, or rewind, or any other tape modes. In essence, this special module would be a tape controller. If the tape has a simple mechanism, this will be a simple module and may be designed so that the command/telemetry module can use many of the same modules. The more complicated tape functions such as counting out blocks, end-of-files, and interrecord gaps should always be performed with software.

3.5 The I/O System

The input/output (I/O) interface of the central processor, as described in the final report of Phase 1 of this project,[1] is summarized in Tables 1, 2, and 3 for a typical mission. The term "input" refers to signals into MULTIPAC and "output" refers to signals coming from MULTIPAC.

A typical mission requires 191 input channels and 126 output channels. The science input lines are doubled in order to be connected through two different registers for path redundancy. Also, there is one output channel added for each serial digital input and each serial command output since these must have additional signals to control the serial transfer.

The multiplexing system which has been devised employs each bit in each of the registers of the machine as a bilevel input channel as well as a bilevel output channel. Input instructions read the 12 interface signals present at the register's input into the register and into an accumulator of the logic unit. Output instructions load the register with 12 output bits, which are then transferred into a 12-bit output holding register. This holding register will keep outputting the 12-bit output information until there is another output instruction to that register. All other instructions use the registers as data scratch and index registers without disturbing the interface signals.

TABLE 1

SCIENCE INTERFACE LINES

Magnetometer:	3 Analog Inputs
	2 Serial Commands
	2 Bilevel Commands
Cosmic Ray Telescope:	4 Analog Inputs
	5 Serial Inputs
	1 Bilevel Input
	4 Bilevel Commands
Plasma Probe:	11 Analog Inputs
	1 Serial Command
	1 Bilevel Command
Radio Propagation:	5 Analog Inputs
	1 Serial Input
	1 Bilevel Command
Neutron Detector:	19 Serial Inputs
	2 Bilevel Commands
VLF Experiments:	6 Analog Inputs
	1 Serial Command
	1 Bilevel Command
Micrometeorite Detector:	2 Analog Inputs
	3 Serial Inputs
	3 Bilevel Inputs
	4 Bilevel Commands
Totals:	31 Analog Inputs
	28 Serial Digital Inputs
	4 Bilevel Inputs
	4 Serial Commands
	15 Bilevel Commands

TABLE 2

ENGINEERING INTERFACE LINES

Orientation Subsystem:	5 Analog Inputs
	5 Bilevel Inputs
	10 Bilevel Commands
Power Subsystem:	14 Analog Inputs
	1 Bilevel Input
	11 Bilevel Commands
RF Subsystem:	15 Analog Inputs
	9 Bilevel Inputs
	10 Bilevel Commands
S/C Support Subsystem:	6 Analog Inputs
	2 Bilevel Inputs
	2 Bilevel Commands
Central Data Engr. Subsystem:	5 Analog Inputs
	3 Bilevel Commands
Other Subsystems:	1 Serial Input
	2 Bilevel Inputs
	3 Serial Commands
	4 Bilevel Commands
Totals:	45 Analog Inputs
	1 Serial Digital Input
	19 Bilevel Inputs
	3 Serial Commands
	40 Bilevel Commands

TABLE 3

I/O CHANNELS REQUIRED

Inputs (Science lines doubled):				
Analog	-	Science	62	
		Engineering	45	
			107	107
Serial	-	Science	56	
		Engineering	1	
			57	57
Bilevel	-	Science	8	
		Engineering	19	27
			191	191
Outputs				
Serial Commands	-	Science	4	
		Engineering	3	
			7	7
Bilevel Commands	-	Science	15	
		Engineering	40	
			55	55
		Control Lines for Serial Inputs	57	57
		Control Lines for Serial Commands	7	7
				126

The I/O interfaces fall into the following categories, and the methods for handling each of them is discussed in detail below.

Inputs:

Bilevel inputs:	Two-state signals which are sampled asychronously at the interface
Serial inputs:	2- to 17-bit words to be transferred serially into the machine
Analog inputs:	0- to 5-volt analog levels to be converted into digital words with up to 8-bit accuracy

Outputs:

Bilevel commands:	Single-bit commands which are held indefinitely as levels at the interface. Pulsed reset signals are also included here, but they are set, then cleared, by the program.
Serial commands:	Commands of two to five bits which are transferred serially to the peripheral device.

3.5.1 Bilevel inputs.-- These are simply levels which must be read by the processor. They are present as inputs to individual stages of the I/O registers and are transferred into the logic unit by an input instruction addressed to the register. The input instruction (see description of INP in Section 7) reads all 12 input lines of the addressed register into the register and sends those input bits specified by the instruction to an accumulator in the logic unit.

3.5.2 Serial inputs.-- Data words longer than a few bits will be transferred in serial across the interface between the experiments and MULTIPAC in order to keep the amount of wire (and hence weight) to a minimum. A serial input data line will be connected to one of the input stages of an I/O Register. An output line from another register (or the same register) will be used to tell the experiment that serial transfer is to occur. The serial data will appear as sequential bilevel inputs to this single input channel of the I/O Register. Each time an input instruction addresses this I/O Register, reading one bit of data, the register will supply a pulse to the experiment to be used to shift the data to the next bit.

3.5.3 Analog inputs.-- A/D conversion is accomplished by the successive approximation method, where processor software is used for the customarily hard-wired conversion logic. Two addressable registers are equipped as D/A converters. Their outputs are fed into a ladder network

and the resultant analog signals distributed through isolating amplifiers to all devices requiring A/D signal conversion. Each such analog signal will be connected to its own comparator, which will also receive the distributed reference signal from the D/A output. The output of the comparator will then be treated as a bilevel input to one stage of an I/O register.

This method of analog-to-digital conversion was chosen rather than the more standard method of multiplexing the analog signals into an analog-to-digital converter primarily to avoid sending low-level analog signals around the spacecraft where they may be susceptible to noise. The extra weight of shielding the wires could not be afforded. This method sends around only one (actually two for redundancy) analog signal to each experiment which, in turn, returns a digital signal. The cost trade-off is a comparator at each experiment versus a switch at the MULTIPAC for each analog line.

To accomplish the conversion, the program sets a ONE in the high order end of the D/A converter, which is a digital number one-half the number range of the register, and produces an analog output equal to one-half of the analog signal range. The comparator response indicates by its output which signal is larger. This is detected by the program through testing the input channel, and the high order bit in the D/A register is left at ONE or set to ZERO according to whether the analog signal is greater or less than half the signal range. The next highest bit is then set to ONE, and the process is repeated to see whether the analog value is greater or less than 1/4 or 3/4, depending on what the first bit was, and so on. After all bits are thus determined, the converter value is read out of the D/A register. The interface required, apart from the distribution of the D/A converter outputs, consists of one input channel per analog signal.

3.5.4 Bilevel command outputs.-- The register module includes an output buffer register which is loaded in parallel from the shifting register using the word clock timing. This is a typical structure throughout the machine. In this case, however, the buffer register will be loaded only upon receipt of an output command. The outputs of this buffer register constitute the bilevel output channels. This arrangement provides both an input and an output channel for each stage of the I/O register, and since the number of each is approximately equal, it effectively doubles the multiplexing.

Since there is no way of outputting data to one channel without affecting others, and no way of transferring the contents of the output buffer register back into the processor to regenerate the bits which should not be changed, it is intended that a copy of the commands be kept in memory. Command routines would operate on the appropriate word in memory to alter the appropriate bits, using masking instructions that leave the other bits unaffected. Then the updated word would be transferred to the register by an I/O instruction.

3.5.5 Serial command outputs.-- Two bilevel channels are required for serial commands; one to switch the peripheral device to its input mode and the second to provide data levels. A shift pulse from the register will be provided each time it receives an I/O instruction.

3.6 External Characteristics

3.6.1 Parts count.-- Table 4 indicates the size, in terms of standard NAND gates, of the different MULTIPAC modules and the proposed partitioning of them. The general level of 100 gates and less than 50 pins per package was an assumption for the design. This level allows more than the selected vendor (Texas Instruments) to respond to the LSI circuit development program for this system. One LSIC where the use of Texas Instruments' capability for very large circuits could be used effectively is the control section of the Logic Unit. Table 4 shows two alternates to the partitioning of the Logic Unit. Alternate 1 uses a large Texas Instruments' LSIC for all the control gating and alternate 2 uses 5 LSIC's for the same amount of logic to keep within the gate and pin limitation. Reliability estimates use alternate 2.

Using Texas Instruments' discretionary wiring approach, a much higher level of integration is possible (see Section 4) than in other LSI techniques where, to keep the pin count low, the density must be lower than elsewhere in the system. Even so, one type, Control 5, needed 61 pins. To get near 40 pins, this circuit would need to be divided into three circuits since the amount of internal connections per gate is very high. One LSIC using this discretionary wiring technique could be used instead of five different types.

Table 5 lists all the LSI circuit types and their usage. Some (e.g., basic shift register) have large usage and others are used only two or three times. The total types needed are 16 (only 13 if the Texas Instruments' control circuit is used for the Logic Unit).

As much as possible, LSI circuits were reused rather than prolificate a new type. This is most apparent in the use of shift registers. The basic shift register without the parallel input gating could have been used in many places, but a new circuit type could have been required.

3.6.2 Power consumption.-- Table 6 indicates the expected power consumption of about 16 watts for the typical system and about 32 watts for the fully expanded system. These figures are essentially dependent on two budgetary estimates: 1 milliwatt per logic gate and 10 milliwatts per interface circuit. The former figure is based upon the power consumption of the Fairchild LPDTμL logic used in the integrated circuit design and other low-power logic in the same general speed/power class. (See Section 4.) The latter estimate is based upon integrated circuit power levels generally and has yet to be verified by specific circuit design.

TABLE 4

PARTITIONING OF THE LSI MULTIPAC DESIGN

	Circuit Type	No. of LSIC's per Module	No. of Gates per LSIC	Required Pins per LSIC	No. of Gates per Module
Logic Unit (Alternate 1)	Basic Register	5	96	42	480
	16-Way Switches	4 typ. 6 max.	78	42	312 typ. 468 max.
	Complete Control		352	72	352
	LSIC's per module:	10 typ. 12 max.	Total gates per module:		1144 typ. 1300 max.
Logic Unit (Alternate 2)	Basic Register	5	96	42	480
	16-Way Switches	4 typ. 6 max.	78	42	312 typ. 468 max.
	Control 1	1	83	34	83
	Control 2	1	88	40	88
	Control 3	1	83	36	83
	Control 4	1	33	40	33
	Control 5	1	65	61	65
	LSIC's per module:	14 typ. 16 max.	Total gates per module:		1144 typ. 1300 max.

TABLE 4.-- Continued

	Circuit Type	No. of LSIC's per Module	No. of Gates per LSIC	Required Pins per LSIC	No. of Gates per Module
I/O Register	Basic Register	1	96	42	96
	Buffer Register	1	61	39	61
	R/M Control	1	52	18	52
	LSIC's per module:	3	Total gates per module:		209
Memory	Basic Register	1	96	42	96
	Buffer Register	1	61	39	61
	R/M Control	1	52	18	52
	Decoder	1	31	33	31
	Bipolar-to-MOS Interface Circuits	3	14	33	--
	MOS-to-Bipolar Interface Circuits	1	12	26	--
	Memory Storage (CMOS)	128	1880	34	--
	LSIC's per module:	8 plus memory storage	Total gates per module:		240 plus 125 special plus memory storage
D/A Register	Basic Register	1	96	42	96
	R/M Control	1	52	18	52
	D/A Switches	1	8	18	--
	Analog Amplifier	1	1	4	--
	LSIC's per module:	4	Total gates per module:		148 plus 9 special

TABLE 4.-- Continued

	Circuit Type	No. of LSIC's per Module	No. of Gates per LSIC	Required Pins per LSIC	No. of Gates per Module
Command Unit	Basic Register	2	96	42	192
	CMD Control	1	115	27	115
	LSIC's per module:	3	Total gates per module:		307
Telemetry Unit	Basic Register	2	96	42	192
	R/M Control	1	52	18	52
	TM Special	1	29	24	29
	LSIC's per module:	4	Total gates per module:		273
Timing Generator	Oscillator	2 IC's	--	3	--
	Squaring Circuit	2 IC's	--	4	--
	Switch	1 IC	4	14	--
	Counter	3	34	8	102
	LSIC's per module:	3 plus 5 IC's	Total gates per module:		102 plus 5 IC's

TABLE 5

QUANITY OF CIRCUITS PER SYSTEM

Circuit	No. of Gates per LSIC	No. of Pins per LSIC	No. of LSIC's per Typ. Sys.	No. of LSIC's per Max. Sys.	Where Used
Basic Shift Register	96	42	56	107	Logic Unit, all Register Types, Memory
R/M Control	52	18	35	76	Memory, I/O Register, D/A Register, Telemetry Unit
Buffer Register	61	39	31	72	Memory, I/O Register
D/A Switches	8	18	2	2	D/A Register
TM Special	29	24	2	2	Telemetry Unit
CMD Control	115	27	2	2	Command Unit
Counter	34	8	3	3	Timing Generator
Memory Storage	1880 (MOS)	34	768	1920	Memory
Decoder	31	33	6	15	Memory
Bipolar-to- MOS Interface Circuits	14 (Special)	33	18	45	Memory
MOS-to-Bipolar Interface Circuits	12 (Special)	26	6	15	Memory
16-Way Switch	78	42	12	30	Logic Unit

TABLE 5.-- Continued

Circuit	No. of Gates per LSIC	No. of Pins per LSIC	No. of LSIC's per Typ. Sys.	No. of LSIC's per Max. Sys.	Where Used
Logic Unit Alternate 1:					
Complete Control	352	72	3	5	Logic Unit Alternate 1
Logic Unit Alternate 2:					
Control 1	83	34	3	5	Logic Unit Alternate 2
Control 2	88	40	3	5	Logic Unit Alternate 2
Control 3	83	36	3	5	Logic Unit Alternate 2
Control 4	33	40	3	5	Logic Unit Alternate 2
Control 5	65	61	3	5	Logic Unit Alternate 2
Integrated Circuits:					
Analog Amplifier	1	4	2	2	D/A Register
Oscillator	2	3	2	2	Timing Generator
Squaring Circuit	2	4	2	2	Timing Generator
Oscillator Switch	1	14	1	1	Timing Generator

TABLE 6

ESTIMATED POWER CONSUMPTION

Typical System:

Module Type	No. of Logic Gates Per Module	No. of Modules per System	No. of Logic Gates Per System
Logic Unit	1144	3	3432
Register	209	25	5225
Memory	240	6	1440
D/A Register	148	2	296
CMD Unit	307	2	614
TM Unit	273	2	446
Timing Generator	102	1	102
			11,755

Internal Power Budgets

Logic (1 mw/gate)	11.755 w
Oscillator and Squaring IC's	0.200 w
D/A Switches and Amplifiers	0.600 w
Memory quiescent power (100 nw/cell	0.015 w
Memory transient power	0.0XX w
Memory Interface circuits (10 mw/individual circuit)	3.240 w
	Total ≈ 15.8 watts

TABLE 6.-- Continued

Maximum System:

Module Type	No. of Logic Gates Per Module	No. of Modules Per System	No. of Logic Gates Per System
Logic Unit	1300	5	6500
Register	209	57	12369
Memory	240	15	3720
D/A Register	148	2	296
CMD Unit	307	2	614
TM Unit	273	2	446
Timing Generator	102	1	102
			23,371

Internal Power Budgets:

Logic (1 mw/gate)	23.371 w
Oscillator and Squaring IC's	0.200 w
D/A Switches and Amplifiers	0.600 w
Memory quiescent power (100 nw/cell)	0.035 w
Memory transient power	0.0XX w
Interface circuits and sence amplifiers (10 mw/individual circuit)	8.1 w
Total ≈	32.3 watts

3.6.3 Speed.-- The clock frequency of 1.0 MHz and the consequent instruction time of 15 microseconds are based on an anticipated propagation delay of about 50 nanoseconds for the LSI gates. This is somewhat better than the Fairchild LPDTμL circuits, which have a typical delay of 65 nanoseconds and a worst case delay of 140 nanoseconds at -55°C. It is felt that this can be achieved for pin-to-pin paths within an LSI circuit considering the smaller internal capacitances and averaging of internal delays.

The longest propagation path is 16 gate delays (see Section 4.1), including the output delay of the transmitting flip-flop, and the preset time of the flip-flop. At 50 nanoseconds per gate the signal requires 800 nanoseconds to propagate and has 200 nanoseconds to spare. This considers all gates to be identical. It may be feasible to include higher powered and, consequently, faster gates at critical points, which could further improve the delay margin.

3.6.4 Volume.-- The packaging of LSI circuits of this general size seems to require about four times the space of 14-lead flat packs. Therefore, using one-quarter the volumetric density (125 LSIC's per lb) as in the integrated circuit MULTIPAC, the volume can be estimated as follows,

$$\text{Typical System:} \quad \frac{953 \text{ LSIC's}}{0.7 \text{ Space Utilization}} \times \frac{1 \text{ lb}}{125 \text{ LSIC's}} = 11 \text{ lbs}$$

$$\text{Maximum System:} \quad \frac{2314 \text{ LSIC's}}{0.7 \text{ Space Utilization}} \times \frac{1 \text{ lb}}{125 \text{ LSIC's}} = 26 \text{ lbs}$$

3.6.5 Weight.-- Since the weight is largely a function of the packaging rather than of the circuit itself, it may be estimated similarly for one-quarter the density (5 LSIC's per cubic inch) of the IC model.

$$\text{Typical System:} \quad \frac{953 \text{ LSIC's}}{0.7 \text{ Space Utilization}} \times \frac{1 \text{ in}^3}{5 \text{ LSIC's}} = 273 \text{ in}^3$$

$$\text{Maximum System:} \quad \frac{2314 \text{ LSIC's}}{0.7 \text{ Space Utilization}} \times \frac{1 \text{ in}^3}{5 \text{ LSIC's}} = 668 \text{ in}^3$$

4.0 LSI CIRCUIT TECHNIQUES

A survey of integrated circuit manufacturers was made during February and March, 1969, to determine a feasible LSI method of implementing this design. Updated designs of the major modules were first worked out to serve as a basis for the choice. Speed considerations demanded a basic logic circuit of no more than 50-nanosecond average propagation time per logic level. The power budget dictates a consumption of no more than 1 milliwatt per gate.

Partitioning the preliminary designs and estimating the quantity of systems to be built led to an estimate of approximately 15 circuit types and procurement quantities on the order of tens to hundreds of each type. This indicates that the chosen LSI medium must lend itself to the procurement of small quantities at a reasonable cost.

The types of circuits encountered in the survey were P-channel MOS, complementary MOS, bipolar TTL or DTL, and complementary bipolar. LSI media ranged from custom design by manual methods to completely automated design from stored circuit libraries. Intermediate methods involved a standard pattern of individual logic circuits already diffused into the silicon to which custom metallization can then be applied. The latter, as applied by Texas Instruments to their series 54L circuits, was judged most practical for the MULTIPAC design within the time frame of this present contract, although as soon as it matures the complementary MOS technique combined with full design automation would also be very desirable for such purposes.

4.1 Speed

Figure 13 shows the critical propagation path of the logic design. It is the path from a register output through the adder in the logic unit and back to that same register that might occur when an MSKR instruction is being performed.

Counting the wired-or in the adder input selection gating as one gate delay, there are 12 gate delays plus one flip-flop. The worst path through the flip-flop is four gate delays, giving a total of 16 gate delays. An arbitrary delay of 50 nanoseconds per gate is chosen as a reasonable speed for LSI circuitry with 1-milliwatt per gate power drain. This power drain per circuit will yield power levels for a typical MULTIPAC in the design goal range of 10 to 20 watts. Sixteen delays at 50 nanoseconds is 800 nanoseconds, which means that MULTIPAC can conservatively operate at a 1-MHz clock rate. (It is clear this will have to be reevaluated when the final circuits are purchased and breadboarded.)

This delay is about the same as that achieved for the integrated circuit design using Fairchild 9040 circuits (see Appendix C of the MULTIPAC Research Report[2]). In the integrated circuit design the maximum total gate and flip-flop delays ranged from 760 to 1041 nanoseconds, depending

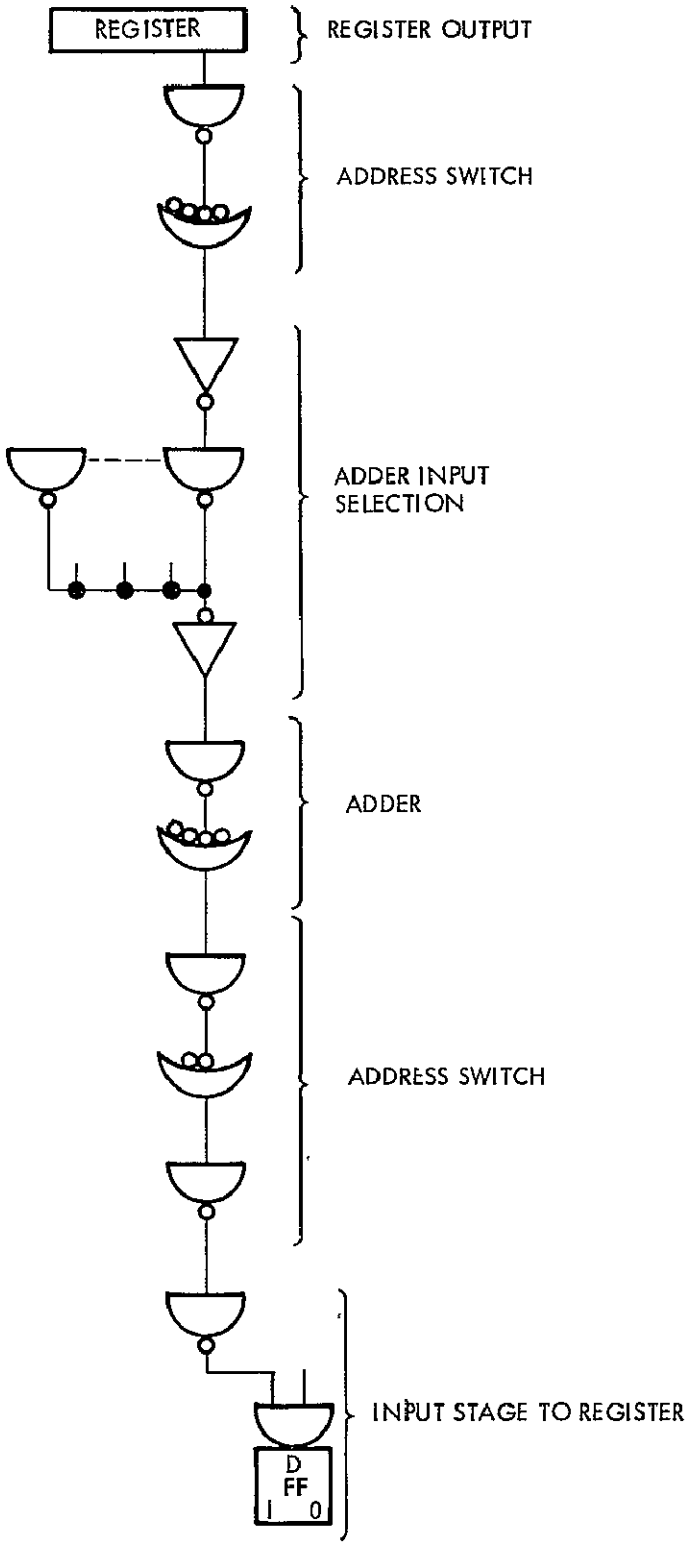


Figure 13. Critical Propagation Path

on temperature. In addition, because D-type flip-flops were not available, a pulse width of 350 nanoseconds had to be added to that delay for the previous design. The delay through a D-type flip-flop is independent of pulse width since it samples the input with the same edge of the clock pulse with which it sets the output. This flip-flop avoids race conditions through internal logical delays (see section 5.1).

4.2 Low-Power Logic Circuits

4.2.1 Low-power bipolar circuits.-- These constitute the low-power, low-speed end of the wide spectrum of bipolar logic circuits on the market. The circuits which are available or proposed in some LSI form are listed below.

	Power Per Gate (mW)	Typical Propagation Delay Per Gate (ns)
Fairchild Low-Powered Micromatrix	3.0	20
Texas Instruments (TI) 54L	1.0	33
Philco Micro-Energy Logic	0.44	50

The propagation times quoted are typical for individual IC packaging and should be somewhat improved on an LSI chip. Note that the circuit which Fairchild intends to market as the low-power entry in their Micromatrix line is not the well-known 9040 series but a faster circuit having three times the speed and power of the latter (nominally 65 nanoseconds and 1.0 milliwatt per gate). Of these, the TI circuits are available in their LSI or MSI format on a custom basis and the Fairchild Low-Powered Micromatrix will be announced within a few months. Philco merely evidences an interest in developing an LSI array using their circuit.

4.2.2 P-channel MOS.-- This is the simplest logic circuit and as such is particularly well suited to LSI. Requiring only a single diffusion and minimal area, this circuit has been the basis for the most fully automated and largest scale fixed pattern integrated circuits.

Part of its simplicity lies in a passive pullup resistor which causes a relatively high power consumption when the circuit is on and a relatively slow rise time when the circuit is turned off. Any attempt to minimize one disadvantage aggravates the other. Dynamic two- or four-phase clock arrangements which switch the power or ground paths minimize or eliminate the period of time for which these load resistors appear across the supply voltage, allowing gate capacitances to hold the data between clock pulses.

The simpler two-phase system, which is the only variation easily implemented in the automated circuit designs, is useful only in relatively slow systems in which the low duty cycle of the clock pulses provides a power saving by enabling the load resistors only a small percentage of the time. The use of LSI's greatly improves the performance of static MOS circuits over their use in individual IC's, however, since node capacitances on the chip are greatly reduced and relatively high-load resistors (75-100 kilohms) can be employed at fair speeds. Some of the speed and power figures given for such devices driving on-the-chip loads in automatically designed LSI circuits follow:

	Power, mW (50% duty cycle)	Average Speed, ns
Fairchild "Micromosaic" Array	0.65 mW	76
American Microsystems, Inc.	1.2 mW	< 100
Texas Instruments	1 mW	100-150

Although within the same order of magnitude, these figures are still not competitive with low-power bipolar circuits in speed-power ratio.

4.2.3 Complementary MOS.-- This circuit type eliminates the long rise time of a P-Channel MOS circuit by replacing its load resistor with a complementary N-channel structure which actively pulls up for any logic condition that does not cause the P-channel structure to pull down. In this sense it is similar to the totem-pole active pullup outputs used in most bipolar TTL circuits and achieves typical propagation delays of 50 nanoseconds even in discrete IC form. Compared to P-channel, the complementary MOS circuit has the disadvantage of requiring a second N-type diffusion into a P-type "tub" and also uses almost twice as many transistors to form the complementary pullup. Thus, it takes the same area as required for bipolar circuits.

Since either the P-channel or N-channel structure is turned off (the other being turned on to one side of the supply voltage), there is no dc path across the supply in the static state except for the leakage of the turned-off MOS devices. This is on the order of several megohms so that there is very little power dissipation in the quiescent state. A dynamic power consumption $\left(P = fCV^2\right)$ is required to charge the node capacitance at the switching rate, which is appreciable. This component is present in the power consumption of any circuit but it is usually negligible compared to the dc component. In complementary MOS, however, if the logic actually switches at the usual clock rates, it is predominant. For the RCA circuits it amounts to 0.6 to 6.0 milliwatts per gate (depending on the particular gate and the capacitive load) at 1.0 MHz. The power consumption in actual use is at least an order of magnitude less, since only a very

small percentage of the circuits in a system will switch on any given clock pulse. This characteristic makes complementary MOS particularly useful for memories since all but an addressed location will hold unchanging information over very long periods of time.

The greatest disadvantage of complementary MOS, at the moment, is that the process has not yet been mastered throughout the industry. At present, RCA is the only manufacturer except for a memory array available from Westinghouse. Several other companies have the process under laboratory study and plan to put such circuits into production within the coming year or so, though none have done so as yet. Typically the first products that are planned are memory arrays. Among the companies planning to enter the complementary MOS field are Fairchild, Texas Instruments, Motorola, Signetics, General Instruments, Radiation, Hughes, Siliconix, and Intersil.

Presently RCA has no LSI vehicle on the market, although a simple array of 48 two-transistor pairs with single level metallization and diffused crossunders will soon be announced. A somewhat more capable array of about the same number of four-transistor pairs and two-level metallization has been developed in their laboratory for NASA Electronics Research Center but it is still considered under development and no firm specifications are obtainable at this time. Its scale of integration is typically a single 4-bit counter on a clip. This is considerably less than the more advanced schemes for either TTL or P-channel MOS. The conclusion reached was that while complementary MOS represents the most favorable circuit type for low-powered logic, it has not reached a sufficient maturity and broad base in the industry to have paired with a suitable LSI vehicle.

4.2.4 Low-power complementary bipolar circuits.-- These are mentioned only for the sake of completeness since they are not presently available as a product. However, there are several development programs looking into this type of circuit which is somewhat similar to the TTL circuit with totem-pole output, except that the internal inversion circuit necessary to drive the pullup transistor is not necessary, thus eliminating several resistors which consume power. Consequently, such circuits have speed characteristics similar to TTL but considerably lower power, though probably not as low as complementary MOS. One of their best uses would, in fact, be to drive large loads of complementary MOS at high speeds with minimal power consumption.

4.3 Methods of Large-Scale Integration

The several methods of providing custom large-scale circuits have various advantages and shortcomings involving the scale of integration desired, the quantity to be procured, the ease of design, turnaround time, and costs. These factors are greatly influenced by the customer service organization the manufacturer sets up to deal with custom requirements and the degree of automation employed to interface his requirements with the IC design and manufacturing processes.

4.3.1 Custom circuits.--

4.3.1.1 Custom circuits made by the manufacturer's standard production methods:-- This method generally costs from $15,000 to $50,000 as a one-time design fee plus $50 to $150 per circuit and is feasible only for large (100,000 or so) quantities. A relatively low initial cost generally means that the manufacturer anticipates writing off the design cost over the total procurement quantity, or expects to be able to market the design commerically. Although it does lead to the most efficient design and layout, this costly method is employed for low quantities only by small companies which do not expect to manufacture a great number of different custom designs or those which have not funded the development of more advanced methods for custom LSI work.

4.3.2 Hybrid packaging.--

4.3.2.1 Hybrid packaging of standard IC's on a substrate containing customized interconnections:-- Actually the IC's so used can be more complex than those which can be packaged in a 14-pin flat pack, but all functionally dictated interconnections must be made to the custom substrate via wire-bonding, flip-chip methods, or beam-lead techniques. This is more of a packaging technique than true LSI and is undesirable for this design because of the indications that reliability is largely a function of the number of interconnections and, hence, the number of monolithic circuits employed in the design, rather than its total complexity. This approach is commercially available, however, in the custom MEMA packages marketed by AMELCO for about $2,500 initial fee and $100-$150 per circuit.

4.3.3 Custom metallization.--

4.3.3.1 Custom metallization of a standard circuit array:-- This is one of the cheaper methods of making customized monolithic circuits, costing between $5,000 and $20,000 for the engineering design of the custom metallization masks. Such a method is normally required for bipolar circuits since their complexity presently prohibits all the masking being laid out by computer. Instead, all the diffusion steps are standard to the array and only the metallization is customized. The Fairchild Micromatrix and what Texas Instruments calls MSI are typical of such methods. Philco evidenced some interest in producing their MEL circuits in such a format. This is also the method used by RCA for their developmental complementary MOS arrays. This method makes relatively inefficient use of silicon area to accommodate the variety of possible interconnections and is consequently limited to about 50-100 gates per chip for reasonable yields. The TI method using more specialized cells offers up to 200 gates equivalent complexity. While single layer metallization is possible in such arrays by using diffused crossunders, the premium placed on silicon "real estate" makes multilevel metallization a practical necessity and those vehicles mentioned use two-level metal except for the simplest RCA array.

4.3.4 Discretionary wiring.-- A very large scale of integration and a high degree of automation are provided in the discretionary wiring of diffused circuit array, the technique which is known by the name of LSI at Texas Instruments. Since the limitation on the size of integrated circuits is the likelihood that a fault will exist somewhere in a large chip area and render the whole chip useless, this method diffuses a very large array of cells over an entire wafer, tests them individually, and interconnects the good ones by a mask that is unique to that wafer.

A cell is defined as the smallest piece of circuitry which is accessible for testing and subsequent interconnection. These vary in size from a single gate to sections of logic of approximately 25-gate complexity. Although commonly used cells are already documented, special ones can be designed for the customer if he has a particularly recurring pattern in his logic. The cells are diffused into the silicon and internally connected by first-level metal in sufficient numbers that the expected yield of each type of cell is more than sufficient to satisfy the expected requirement. The cells are then tested by computer, the wafer characterized by the good circuits it contains and put in inventory. When the wafer is subsequently committed to a requirement, masks for two more layers of metallization are generated by the computer to satisfy the interconnection list supplied.

This method permits logic structures from 200-gate to approximately 2000-gate complexity to be put on a single monolithic wafer (for functions requiring less than 2000 gates, TI used the fixed metallization approach, which they call MSI). The discretionary wiring method was the most advanced found to be available for complex bipolar circuits and is among the least expensive for large devices in quantities. TI quotes a price of $10,000 initial fee plus $2,000 each for a minimum quantity of five for these circuits which are better than ten times more complex than any other bipolar array.

4.3.5 P-channel MOS technology.-- The most fully automated custom LSI methods were found in the P-channel MOS technology. Similar methods exist at many companies including Fairchild, Texas Instruments, American Micro-Systems, and General Instruments. A typical system of this type incorporates a circuit family or library, the diffusion masks for which are stored on a computer tape. The design is accomplished using this library. A list of the circuits used and their interconnections is then prepared in a specific format. This list is used as input to the automated design program which calls out the cells, places them on the chip for optimized interconnection routing, and makes the diffusion and interconnecting metalization masks automatically. Thus, if the design is specified by the customer in the form of a card deck having the appropriate format, the chip design and layout can be wholly automatic (though manual intervention at some stages is also allowed). The design thus specified can also serve as the input to a logic simulation program to assure the customer that his design is functionally correct and it can also be used to generate an optimum test sequence for the finished product. Unfortunately, this approach is not applicable to the present design since it is available only

for P-channel MOS circuits, which are too slow for this purpose. It seems likely that complementary MOS designs will eventually be implemented in this fashion, and this would be a highly desirable combination for spacecraft logic.

4.4 Memory Circuits

The memory storage medium postulated is a modification of the 256-bit complementary MOS memory chip under development for NASA by Westinghouse on Contract No. NAS-5-10243.[2] As with the logic circuits, the chief reason for this choice was the low power drain of this medium. In a flip-flop memory, one important consideration is the power required to maintain the memory contents under dc conditions. Standby power required for this MOS circuitry is reported as typically less than 100 nanowatts per cell, which would be approximately 2.5 milliwatts for a 2048-word, 12-bit memory. This is negligible compared to the operating power of the one chip in the memory selected by the addressing logic, which is reported as 30 milliwatts for the present device containing 16 words of 16 bits each when cycled at a 0.5-MHz rate. If this were scaled to a 12-bit word with a 15-microsecond cycle time, the indications are that the 2048-word memory would consume on the order of 3 milliwatts. This power is, in turn, negligible compared to that required by the bipolar portions of the memory module, especially the circuits interfacing the MOS chip. These would probably require special design to minimize their power consumption.

In addition to shortening the word length to 12-bits for the present system (or removing power from those bits not needed), two other changes in the Westinghouse chip are needed to make its use practical in MULTIPAC system. There are presently no clear provisions for the use of multiple 16-word chips in a larger memory. There is no method of expanding the addressing structure except for gating the "strobe" and "write" signals of each chip separately, and the "bit lines" are not presently such that they can be collector ORed to form the larger memory. If both these signals from each chip have to be separately ORed or gated after leaving the chip, it would require more interface circuits than memory chips and the power demands would be exhorbitant. Making these changes on the chip would seem relatively easy, however. The change to the output circuits to permit ORing them means gating out the active pull-up circuit as well as the pull-down circuit from non-selected chips in a similar manner to what is now done for writing. This leaves only the output circuits from the selected chip active to drive the bit lines. In write mode they too are disabled, leaving only the write drivers active on these lines. In addition, the addressing logic should be expanded to include two more inputs which would gate all address decoders and appear on outside pins to be used for x-y coordinate addressing of the chip. This would reduce the number of interface address drivers required from 128 to 32 for a 4096-word memory, with a corresponding saving in power. The logic diagram for the memory assumes these modifications will be made.

4.5 Special Circuits

There are several non-logical circuits required which should be especially designed for MULTIPAC, such as the memory interface circuits mentioned above. Such circuits are not a part of the logic design but they do influence the parts count, power and reliability estimates. Therefore, certain assumptions have been made concerning them as follows:

(1) Clock generator: This circuit must produce a 5-volt square wave at 1.0 MHz. Two circuits, oscillator and squaring circuit assumed, with 100-milliwatt power consumption.

(2) Bipolar-to-MOS interface circuit (BMIC) which interfaces the MULTIPAC logic levels to the memory. Two circuits per IC package assumed, with 10-milliwatt power consumption per circuit.

(3) MOS-to-bipolar interface circuit (MBIC) which interfaces memory to the MULTIPAC logic. Two circuits per package assumed, with 10-milliwatt power consumption per circuit.

(4) Analog switch (ANSW): This circuit switches D/A inputs to ladder network. Eight circuits per package assumed, with 25-milliwatt power consumption per circuit.

(5) Analog amplifier (ANAMP): These are isolation amplifiers which supply the analog output to the experiments. A standard integrated operational amplifier circuit could be used. One circuit per package assumed, with 100-milliwatt power consumption.

4.6 Circuit Choice

In spite of the frequent and casual use of the term in the industry, true large-scale integration still seems to be in its infancy. Only the largest manufacturers (and some MOS specialists) have any well-formulated means of responding to customer requirements, and even these are still in current development. Present yield considerations limit fixed-wired chips to 100-120 mils square, and the most useful method of tailoring this area to perform a complex custom requirement is in automated P-channel MOS design, but these circuits are too slow for the MULTIPAC requirement.

For MULTIPAC, the recommended technique is the bipolar circuits using the TI method of discretionary wiring, which overcomes the limitation on chip size. Since each wafer produced by this method requires unique masking, it is essentially a low-volume process and is priced as such, in keeping with the MULTIPAC requirements. Additionally, TI has a version of the more limited fixed-wired technique which could be used for the smaller, repetitive, and, hence, higher volume, sections of the MULTIPAC design, such as the basic register circuit (see Figure 15).

Complementary MOS as a circuit type is well suited in speed and power to the MULTIPAC requirement, but at present there is no LSI vehicle for it that is adequate and sufficiently well defined to permit its being used as the basis for a design. Memory chips are presently available from two companies, however, with several more companies planning such products for the immediate future, and so its use in this design as the memory medium does appear feasible. As soon as the larger IC manufacturers have established an IC line in complementary MOS, the circuit will no doubt appear in their automated design LSI formats and will probably find a large usage in spaceborne systems such as this one, but such developments cannot be anticipated within the time frame of this contract.

5.0 DETAILED DESCRIPTION OF MODULES

All data transfers between modules are serial. Synchronous machine timing is provided by two clock signals, the shift clock (SC) and the word strobe (WS). These are generated in the timing generator and distributed to all modules by triplicated signals driving majority voting gates at each module interface. Timing consists of 14 SC pulses followed by one WS pulse at equally spaced intervals of approximately 1 microsecond. The machine cycle is, therefore, about 15 microseconds. (Figure 26, timing diagram of the register and memory units, shows these signals.) Since the ac flip-flop design used triggers on the rising signal edge, the clock pulses are negative going to provide trailing edge triggering and thus permit the gating of clock pulses. Except for such dc gating of clock pulses, the pulse width is irrelevant. For convenience, the clock pulse width is assumed to be 50 percent of the period.

The actual 12-bit data transfer is preceded by the transfer of a 2-bit control code on the same line. It is by the transfer of this code that the logic unit controls the operations of the other modules with which it communicates.

The design described in detail in this section is complete but has not been breadboarded; consequently, it may have errors. The logic symbols used follow MIL STD 806B per paragraph 4.3 (Basic Logic Diagram without physical implementation). The discussion which follows is detailed enough to replace the usual logic equations. Logic equations are usually difficult to follow, whereas the discussion leads the reader through the logic in proper order and supplies the reasons behind the chosen implementation. Logic designs may be simulated by a computer without equations (see Appendix B).

5.1 Flip-Flops

In the design of an LSI system, one is not limited to the "off-the-shelf" circuits. For example, flip-flops have only those gating networks on the input which are actually used.

For MULTIPAC we have designed three different flip-flops: (1) a set-reset flip-flop without internal delay, (2) an AND input delay-type flip-flop (D flip-flop) and (3) a NAND input delay-type flip-flop. These are shown in Figures 14, 15, and 16, with symbols on the left and logic implementation on the right.

The set-reset flip-flops set the output to a ONE if the SET input is a ONE and to a ZERO if the RESET input is a ONE when the clock (CLK) is high. If both SET and REST are ONES, the final state of the flip-flop is indeterminate. The setting is accomplished by directly forcing one of the two cross-coupled flip-flop NAND gates to a ONE output. Since there is no delaying action, the flip-flop's outputs cannot feed its own inputs or that of any similar flip-flop from the same clock.

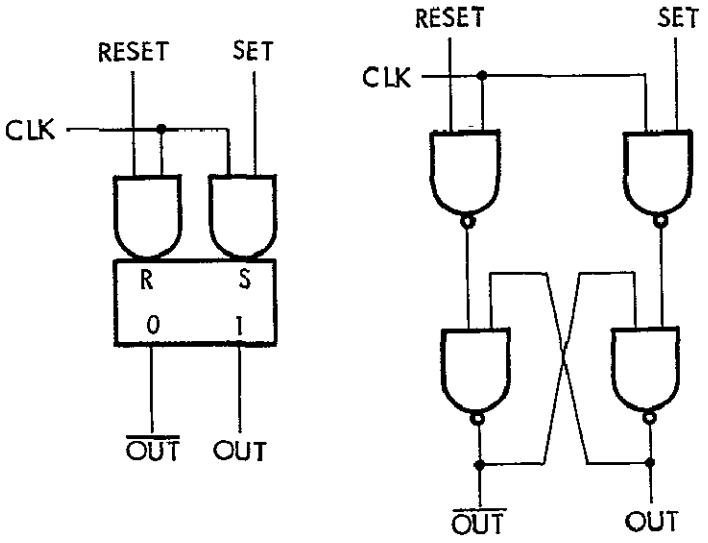


Figure 14. Set - Reset Flip-Flop

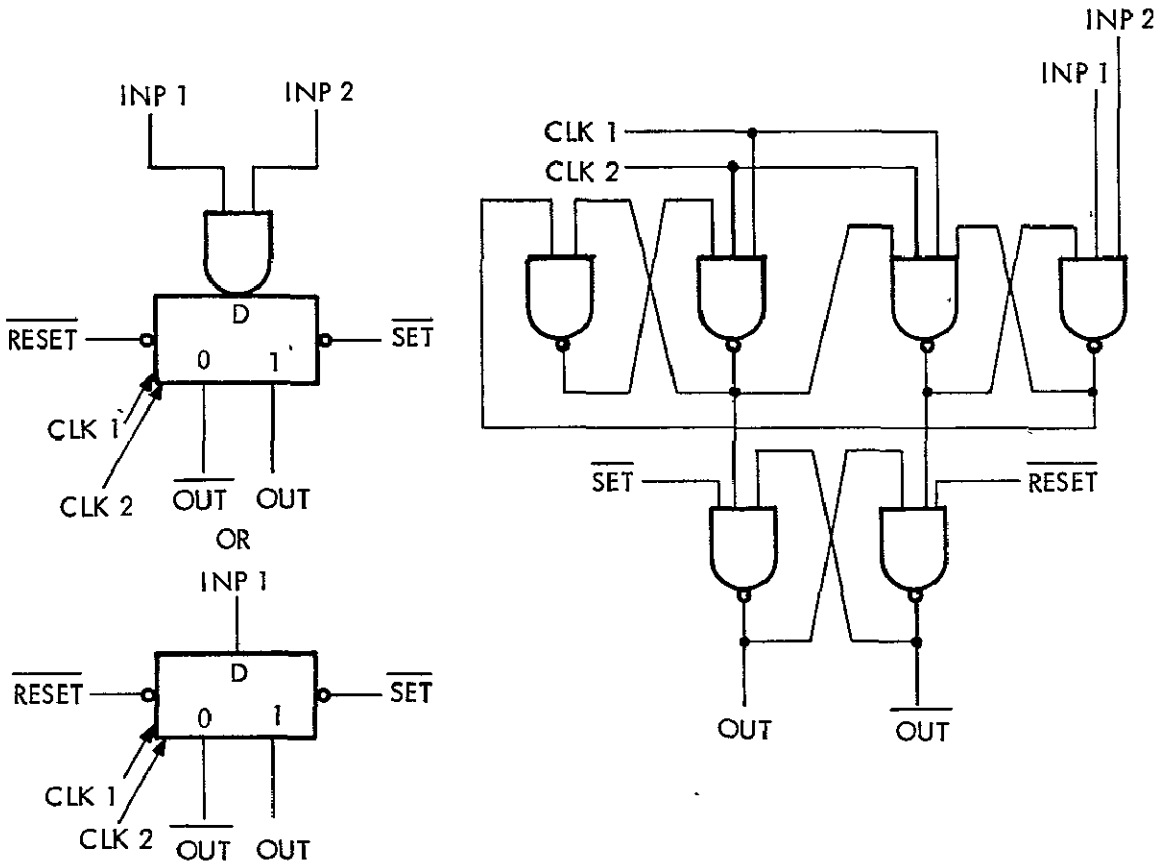


Figure 15. AND Input D Flip-Flop

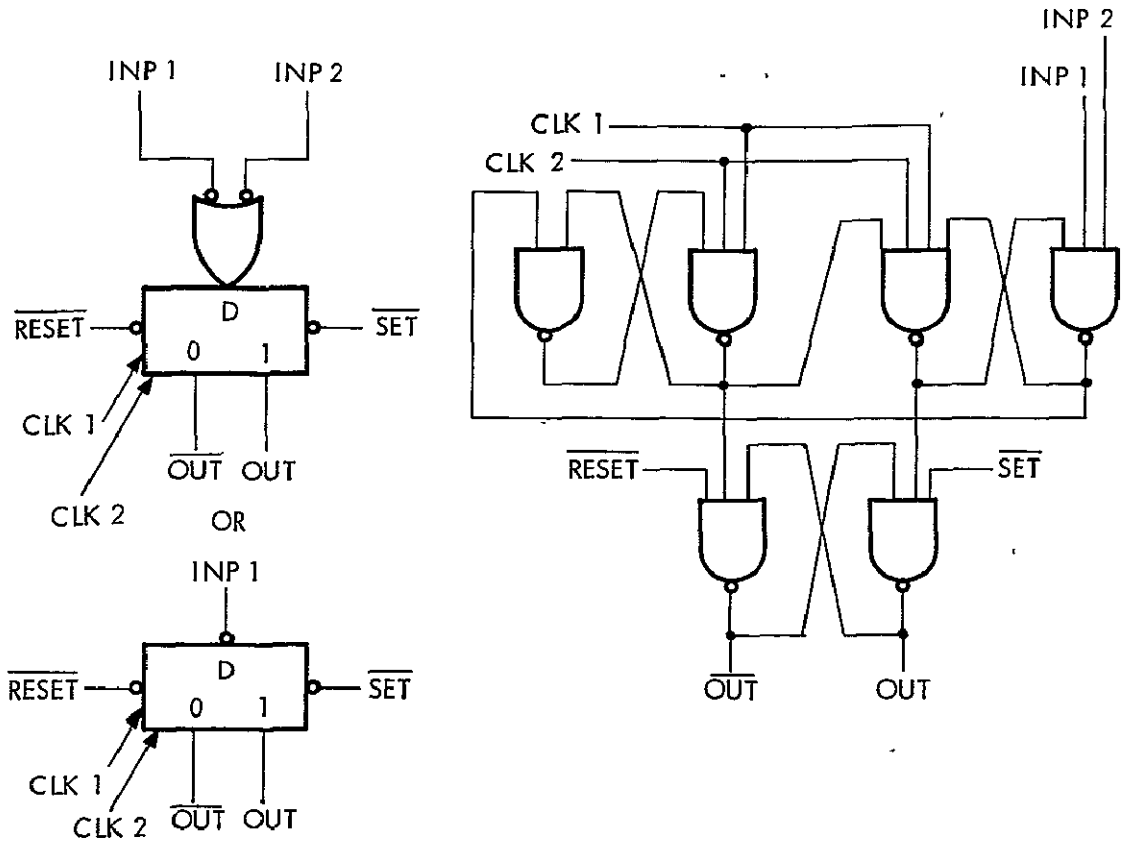


Figure 16. NAND Input D Flip-Flop

The AND input D flip-flop sets the output to the logical AND of its inputs when the clock goes high. While both clocks (CLK1 and CLK2) are high, the inputs are prevented from affecting the outputs. While the clock is low, the output flip-flop (formed by cross-coupled NAND gates) are isolated from the inputs.

The action of this AND D flip-flop is as follows. With both the clocks low, the two NAND gates with clocks as inputs will have ONE outputs no matter what the other inputs to these gates are. The NAND gate with the flip-flop input signals will form the NAND of the inputs, and the gate on the far left of the diagram will invert this, forming the AND of the inputs. When both clocks go high, the middle gates are enabled and the NAND and AND of the INP1 and INP2 information are forced into the output flip-flop. At the same time, this NAND and AND information is stored in two flip-flops formed by pairs of the top four gates. If, then, the inputs change, they will not be able to change the middle two gates. For the logical case when the third input to the NAND gate with INP1 and INP2 is a ZERO, no change of states of INP1 or INP2 can change that NAND gate and, hence, the outputs. If that third input to the input NAND was a ONE, then the NAND gate for which it is an output will have a ZERO input from the top left flip-flop, preventing any affect on it from changes of the input NAND circuit. For this same reason the top left NAND also cannot change. Thus, the internal logic of the flip-flop prevents changes on the inputs to affect the output except at the time when the clock goes high after being low long enough for gates to stabilize.

The NAND type input D flip-flop is identical to the AND D flip-flop except that the labelling of the outputs is reversed. This has the effect of forming a NAND of the inputs.

5.2 Basic Register Circuit

All shift registers in MULTIPAC may be made of the single shift register LSI circuit shown in Figure 17.

This basic circuit will shift right when the SHIFT line is high and a clock pulse occurs. When the INPUT line is high and a clock pulse occurs, 12 inputs (labelled $\overline{\text{IN0}}$ through $\overline{\text{IN11}}$) will be clocked into the register. Since there is an inversion in the input gating, the inverse of the input signals is transferred.

If the inputs are each connected to the next stage to the right, then the register can shift left as shown in Figure 18. This connection results in a LEFT/RIGHT shift register, where the INPUT line is a LEFT SHIFT line, and the SHIFT line is really a RIGHT SHIFT line. The IN0 line becomes a RIGHT DATA IN and the DATA IN a LEFT DATA IN. These connections are used for the two accumulators of the logic unit.

5.3 16-Way Switch Circuit

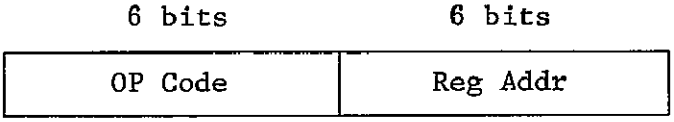
Another generally used LSI circuit is the 16-way switch shown in Figure 19. This circuit is used in the logic unit for switching data to and from memories and registers.

Sixteen of the NAND gates decode one out of 16 states of the four input levels. Each of these decoded outputs feeds 32 NAND gates which gate in and out 16 ways. The data gates also have signals for enabling all input gates or all output gates. There is an input enable (INPUT SELECT) and an output enable (OUTPUT SELECT). To keep the pin count to a minimum, the four register inputs to be decoded are brought in single-ended and inverted internally and the 16 gated data signals are ORed internally.

5.4 The Logic Unit

The Logic Unit (Figure 20; see also Appendix C) is connected to all other modules in the system and controls those which it addresses. At any one time, a module is addressed by only one logic unit, the one to whose process it is assigned. (Use of a module by more than one processor for purposes of intercommunication must be coordinated between the two programs concerned.) The logic unit selects one memory as its source of program and another (although it may also be the same one) as its source of data locations by means of two 4-bit page registers (PMR and DMR).

5.4.1 Instruction decoding.-- The MULTIPAC instruction word usually has the form



where the OP field determines the operation to perform and the R field refers to either a register to operate on or a register to use for indexing. In this latter case, the next word following is used for the address field, requiring two MULTIPAC words for memory reference instructions.

In general, the OP field is further divided as shown at the bottom of Figure 21 into a 4-bit OPC field, a 1-bit M field, and a 1-bit A field. The A field specifies which of two accumulators to use and the M field specifies whether or not memory reference (and, hence, a one- or two-word instruction) is needed.

Figure 21 shows the Boolean truth table from which MULTIPAC was designed. A few exceptions to the above generality can be noted, and each of these are decoded separately at appropriate places in the Logic Unit. Two M = 1 instructions, MDI and JMPR, are single-word, nonmemory reference instructions which act very much like memory reference instructions. One M = 0 instruction, INP, is a two-word instruction which uses the second word as a mask. It is coded here since it is primarily a register operation.

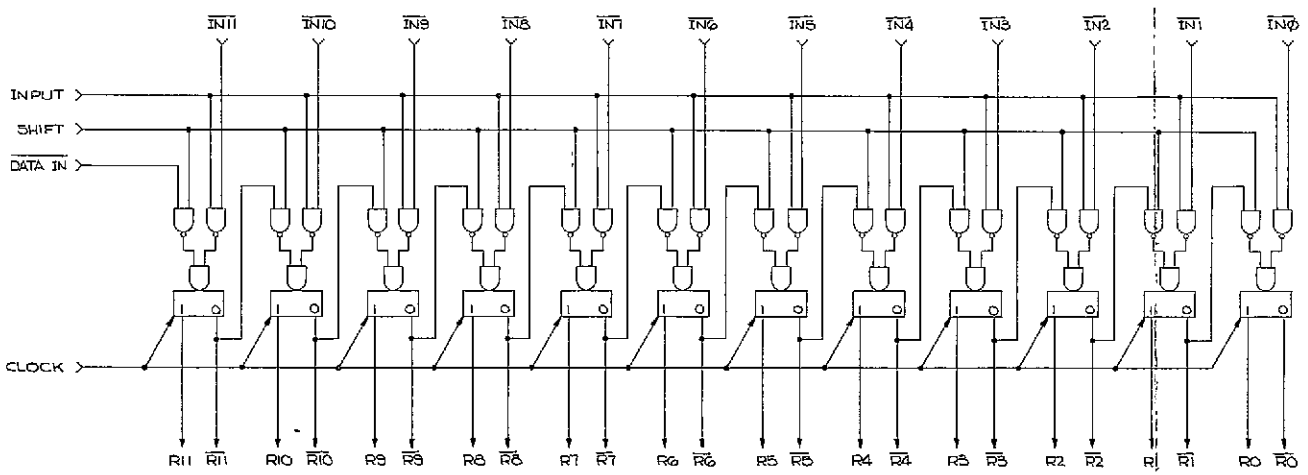


Figure 17. Basic Register

A

FOLDOUT FRAME 1

B

FOLDOUT FRAME 2

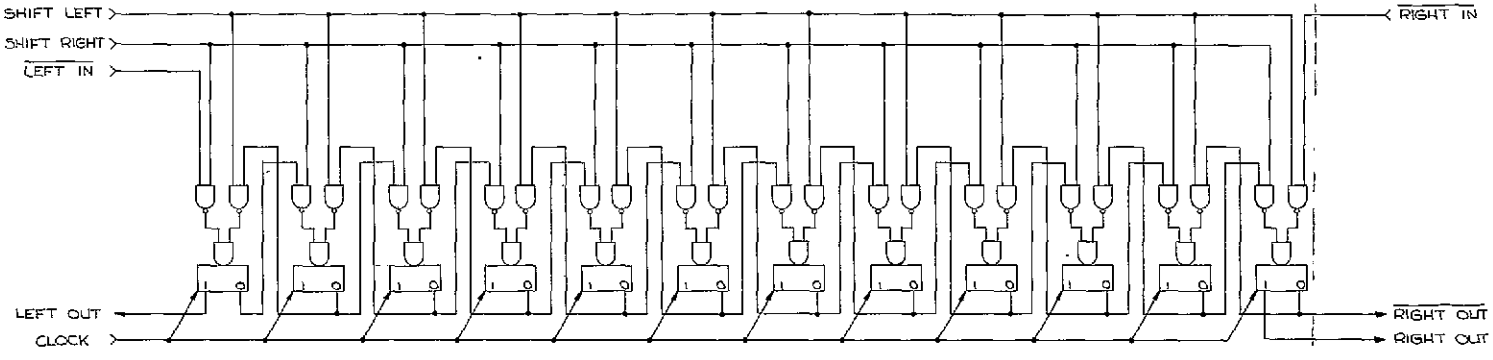


Figure 18. Basic Register Connected as Left/Right Shifting Register

A

FOLDOUT FRAME 1

B

FOLDOUT FRAME 2

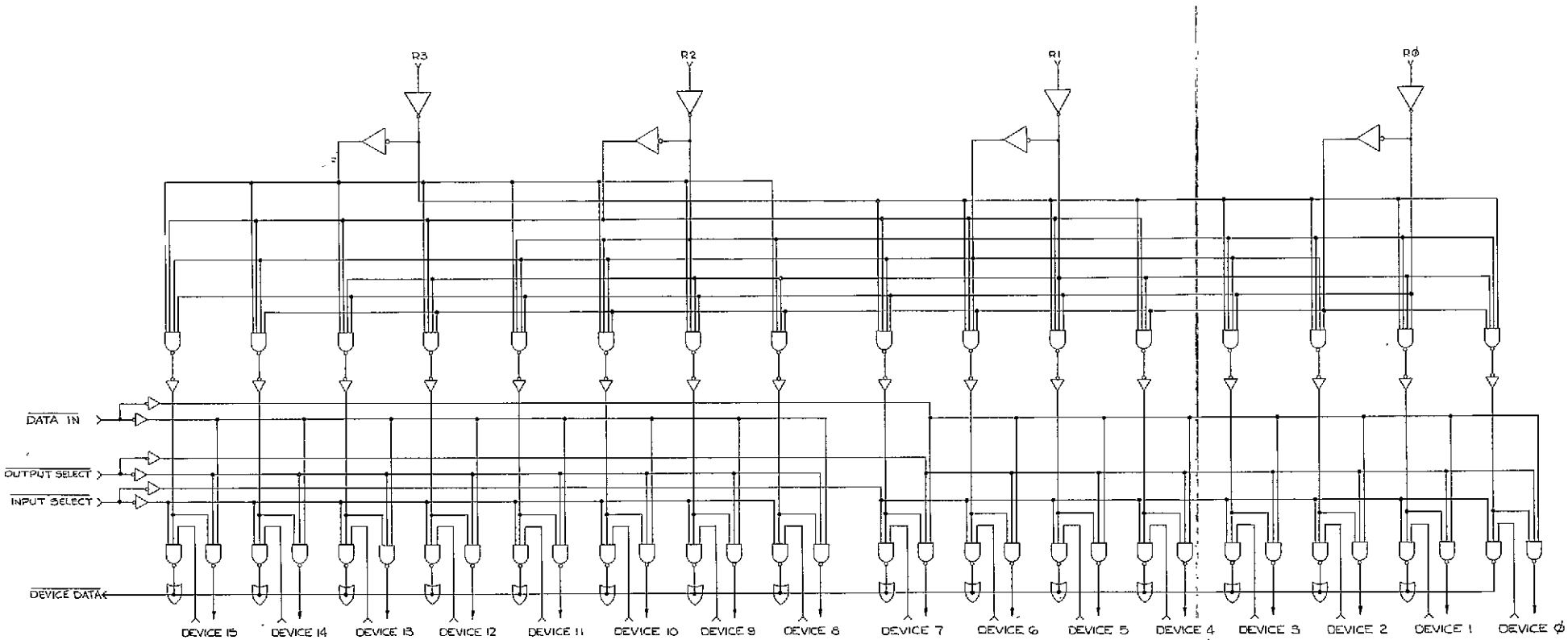


Figure 19. 16-Way Switch

A

FOLDOUT FRAME 1

B

FOLDOUT FRAME 2

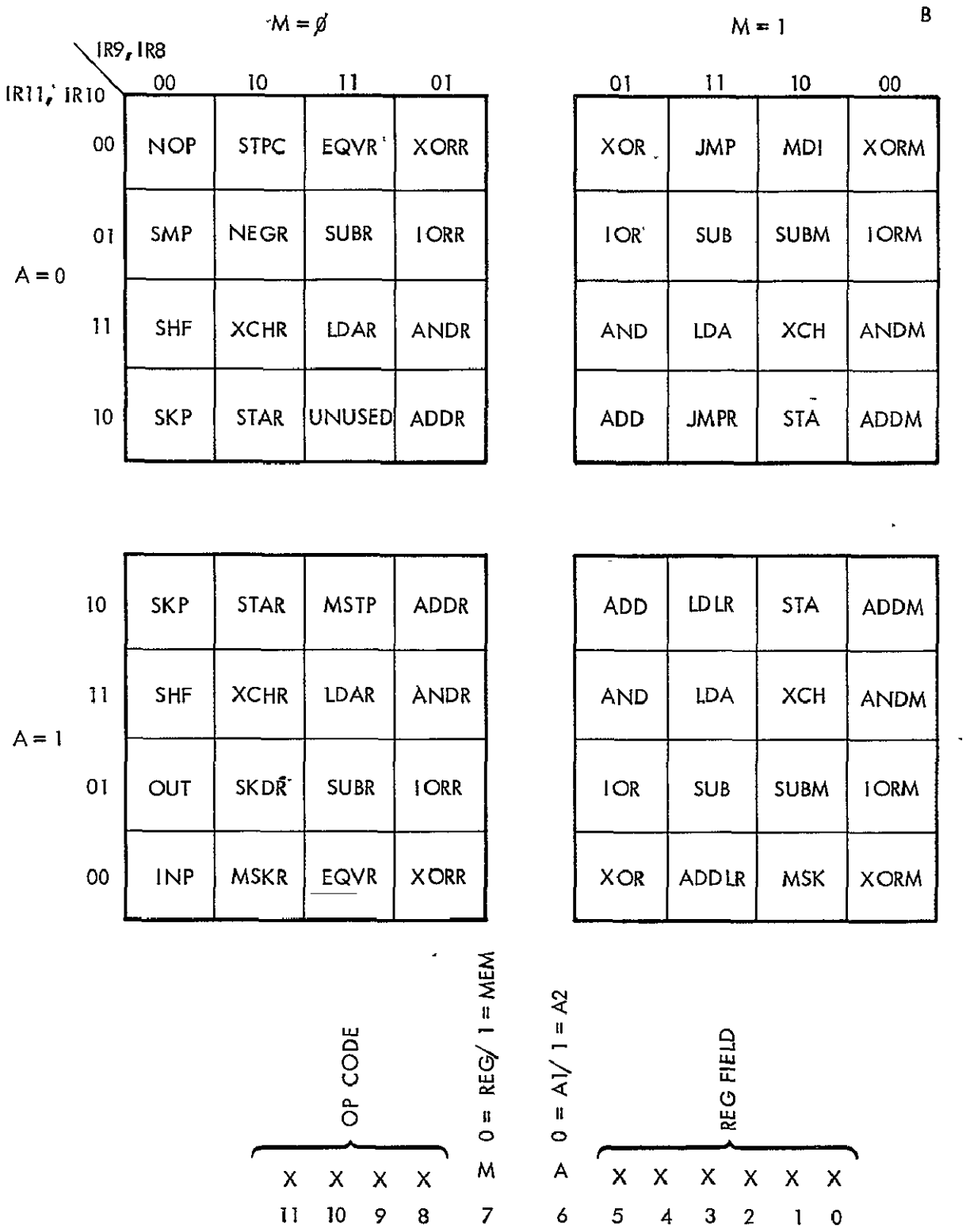


Figure 21. LSI MULTIPAC Operation Codes

SMP, SHF and SKP are exceptions to most of the other instructions, and the effect of decoding them appears many places in the logic unit. SHF and SKP use the R field for further decoding of the instructions, as shown in Figures 22 and 23. For the shift (SHF) instructions, the R field specifies amount, direction and control (e.g., arithmetic, logical, double length, etc.) of the shift. For the skip (SKP) instruction, this field determines the condition. For SMP, the R field is used for the value to store in either data or program memory page registers.

5.4.2 The control codes.-- Control codes are generated by a 2-bit shift register called the Control Code Register, which is set by the word strobe according to the next operation to be performed and shifts this 2-bit code out through the adder to the registers and the memory units. Information for setting the control codes is, as a rule, decoded from the instruction shift register so that the same word strobe which dumps the instruction into the instruction buffer register also sets the codes simultaneously in anticipation of that instruction. The four codes and their effect in the register or the memory control unit can be seen by reference to the dual-purpose register and memory control unit module. Code 00 is treated as a No-op, 10 shifts the register or memory register, 11 results in an output instruction or memory read, and 01 results in input instruction or memory write. Thus, for the register instructions, the usual code is 10, causing the register to shift.

The control codes for various instructions are shown in Table 7. The first (rightmost) bit is made a ONE for input (INP) and output (OUT) instructions only; the second bit is made a ONE for all instructions except NOP, SHF and SKP instructions which do not shift the register and INP, which inputs the register. All of these fall in the class of nonindexed instructions. If the next instruction is a memory instruction, however, it will first call for an index cycle. In that case, the code generated is 11, the code that will cause a memory read. At the same time, however, the index register must be made to cycle, the code for that being 10. Therefore, 11 is generated in the control code register, but one bit of it is blocked on the bus leading to the registers.

The program memory must also be controlled by such control codes and, in general, the first two bits are forced to a ONE by hard-wired logic. This is the read code and causes the program memory constantly to read. An exception occurs here, however, when the program memory and data memory are the same. In this case, a level from the comparator determining this blocks the signal, forcing it to a 00 during the execute part of the cycle. Since the program memory bus and the data memory bus OR together, and the program memory bus is forced to ZERO, the program memory can respond to the code on the data memory bus and will either write or read as the instruction being executed indicates.

5.4.3 The sequence counter.-- The sequence counter is a two-stage shift counter (SQ0 and SQ1) which governs the cycles of any particular instruction. SQ1 shifts into SQ0 whose inverse then shifts into SQ1 This

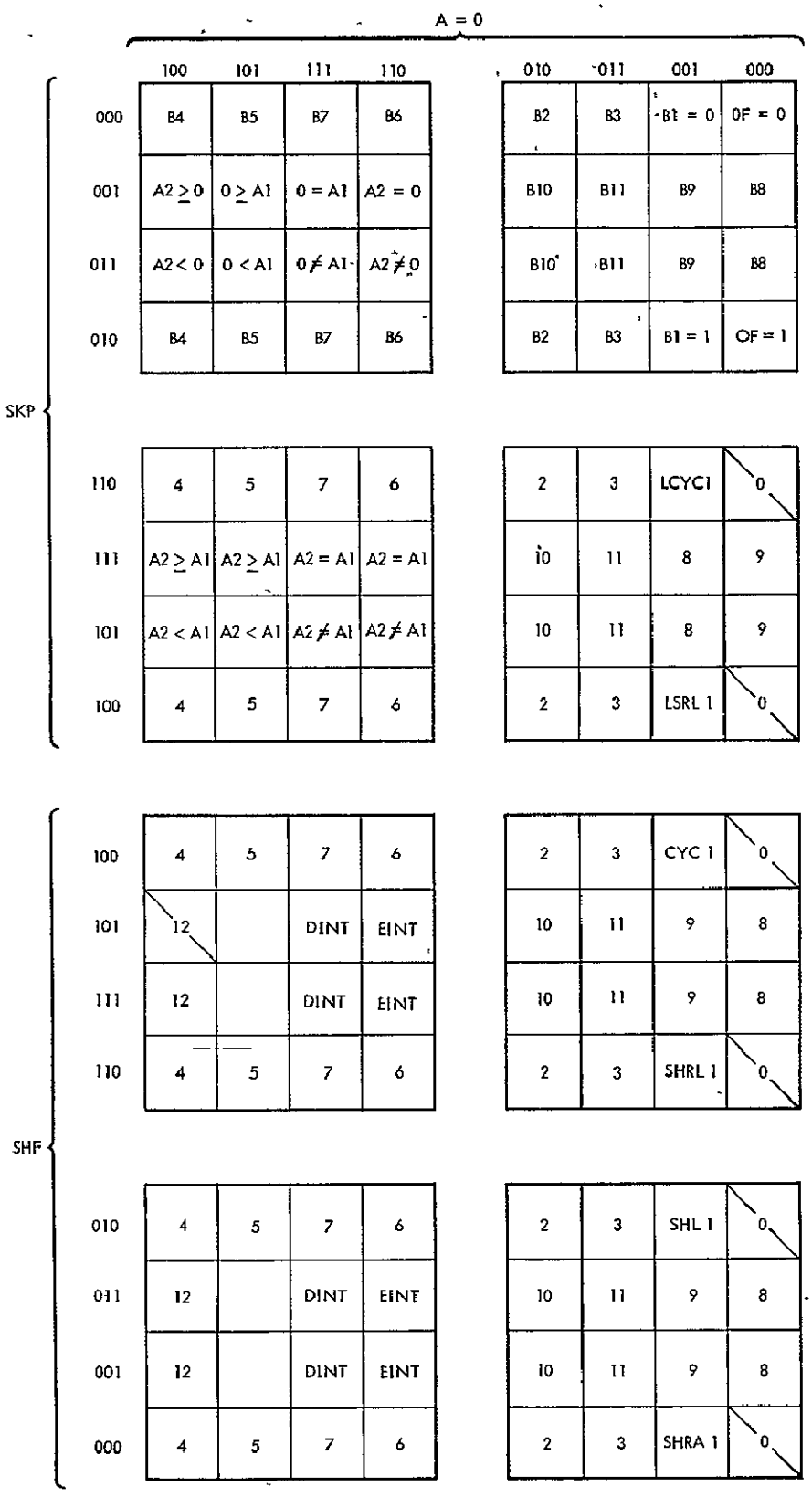


Figure 22. R Field Coding for SHF and SKP, Part 1

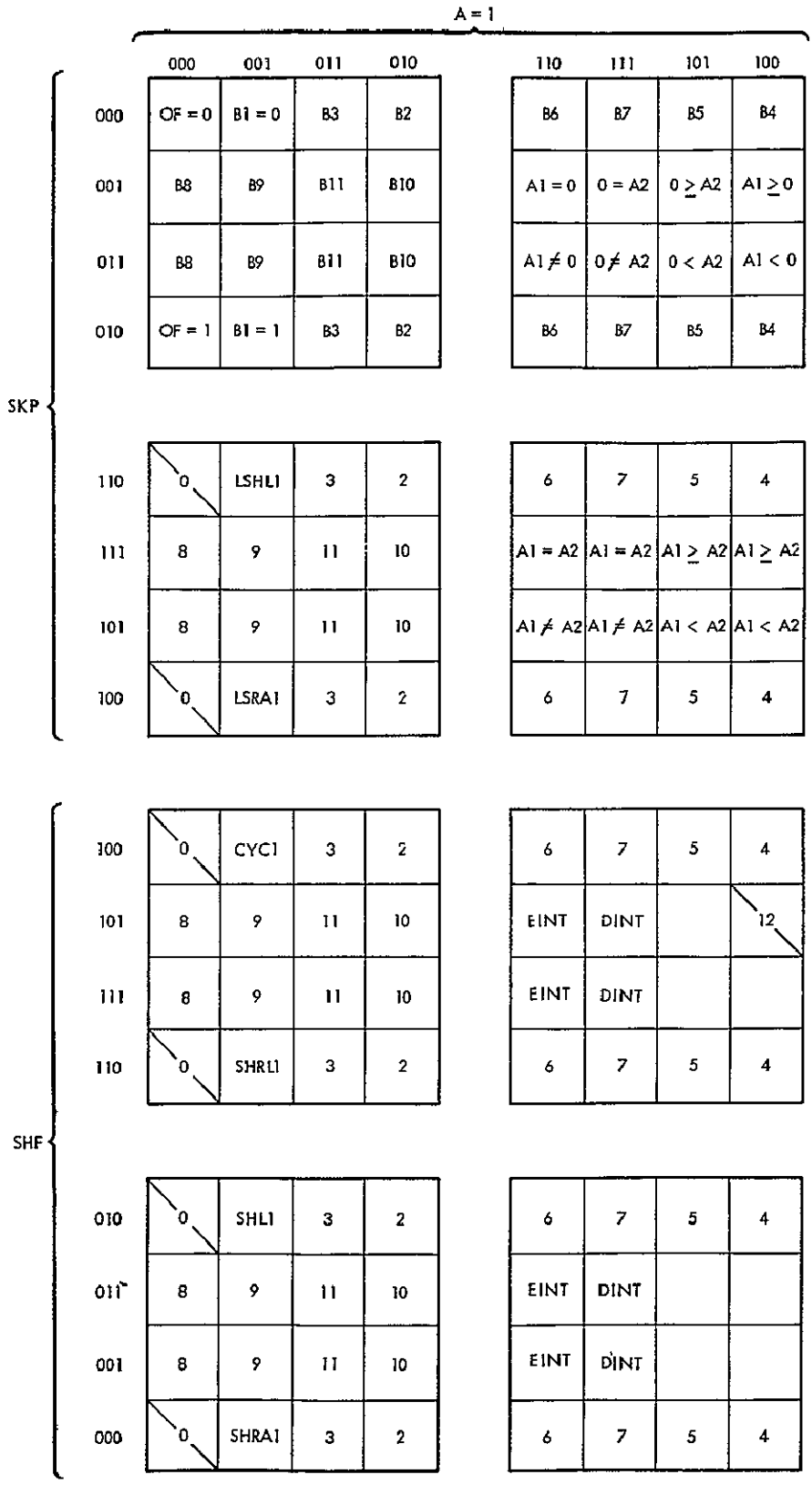


Figure 23. R Field Coding for SHF and SKP, Part 2

TABLE 7

CONTROL CODES

MACHINE STATE	INSTRUCTION DECODING	REG CODE	DM CODE	PM CODE
CNI	ISR7 = 1	10	11	11
CNI	ISR7 = 0:			
	SKP	00	00	11
	SHF	00	00	11
	CNTRL	00	00	11
	NOP	00	00	11
	OUT	11	00	11
	INP	01	00	11
	OTHERS	10	00	11
$\overline{(CNI)}(IND)$	IR8 = 1:			
	JMP	00	00	11
	JMPR	00	00	11
	LDLR	00	00	11
	ADDLR	00	00	11
	OTHERS	00	10	11
	IR8 = 0:	00	00	11
$\overline{(CNI)}\overline{(IND)}$	ALL	00	00	11

is the pattern of the shift counter. Thus, it, in general, steps through the following states: 01, 00, 10, and 11. These may be given the four functional titles: Index, Execute, No-op, and No-op. At some point in this cycle, the condition is detected that the present instruction is completed, and at this point a copy next instruction (CNI) signal is generated. This resets the sequence counter to either the index or execute state, depending on the next instruction. In which of these two states the next instruction commences is determined by bit 7 of the next instruction decoded from the instruction shift register since that will be the location of the next instruction at that time.

If bit 7 of the instruction to be executed is a ONE, it indicates an instruction which must be indexed and, therefore, the counter will be reset to state 01, or the index state. If bit 7 of the instruction is ZERO, the reset state will be 00, or the execute state. Having thus begun in either the index or execute state, the only remaining variable in the cycle is with what state to terminate the instruction.

The only instruction to terminate after the index cycle is the modify instruction (MDI) command. This is decoded with the index state 01 to produce the CNI signal. Most commonly, the CNI signal will be produced by the next state (00 or execute) provided that one of three ORed signals is present. One of the ORed signals represents the lack of comparison between the program memory register and the data memory register (i.e., the program memory and data memory are not the same).

A second condition that will terminate the instruction at this point is an OR of the jump to register contents (JMPR), load register with literal (LDLR), and add literal to register (ADLR) instructions because they require no further machine cycles even if the program memory and the data memory are the same.

The third condition which terminates the instruction is if bit 7 is a ZERO, i.e., if the instruction falls in the category of single-cycle instructions, one single-cycle instruction, skip, is only conditionally single-cycle. There is an additional inhibit generated by the skip instruction and the test condition which will inhibit CNI from being generated in the execute cycle, and, therefore, the sequencer will go on to further states.

The next two states, 10 and 11 of the sequence counter, are both no operation (NOP) states. They are used for conditions when program memory and data memory are not the same, in which case a certain time must be consumed (one or two cycles to allow memory operation) or they provide the two-cycle skip generated by the skip instruction.

If the program memory and the data memory are the same, three cycles are required for instructions which do not write back in memory. These are coded with a ZERO in bit 8, and upon detecting this during the first of the two NOP states, the CNI signal will be generated. If bit 8 is a ONE, the instruction is one which writes back in memory. When data are in the program memory, instructions require four cycles and so the sequencer

steps further to the second of the two NOP states. In this final sequencer state of 11, the CNI signal, which restarts a new sequence, is always generated.

5.4.4 Instruction timing.-- The MULTIPAC processor operates with overlapped instructions such as those indicated in Table 8. In this table each entry represents one cycle time of the machine, including a two-bit control code time and a 12-bit shift. The slashes are used to indicate the contents shifting into a register on the left and the contents shifting out of it on the right of the slash. The program counter increments regularly, and at the same time its incoming contents are transferred into the program memory data register with a control code, causing a read. The memory reads at the first part of the cycle during the 2-bit control code time, and the contents of that location can be read out in the 12-bit shift which follows. This word is shifted during that time into the instruction shift register and, with the following word strobe, is transferred into the instruction register where it is decoded. Note that this requires a memory read time of less than 2 microseconds (2 shift pulses).

Certain instructions have been assumed in the figure to give an example of the overlap and the internal timing of certain instructions. The first instruction for address ZERO is assumed to be an add and store (ADDM) indexed by register R. Thus, the sequencer will be initiated in an index cycle. This will cause the contents then emerging from the program memory, namely, the contents of location 1, to be added to the contents of register R and transferred into the data register of memory 2 as the effective data address. The code used is a read, and one cycle later, the contents of that effective address are available to be shifted out of the memory register. In this next cycle the sequence counter has advanced to the execute state, and during this cycle, the contents of the effective address are summed with the contents of the accumulator and transferred to the accumulator and back into the memory register with a write code. While this operation is being done, the next instruction is shifting into the instruction shift register so that with the next word strobe, it is transferred into the instruction buffer register. Since this next instruction is not an indexed instruction-in-the example given, the sequencer started in the execute state. The assumed instruction is a skip illustrating the use of SKP and jump (JMP) together. Accumulator 2 is subtracted from Accumulator 1 and the sign determines whether or not the skip occurs. The example assumes that the skip does not occur in order to demonstrate the jump; however, the two NOP's shown in parentheses indicate what will happen if the skip is effective. Assuming this skip is not effective, the next instruction is a jump, causing a typical index operation. The index results are shifted into the program counter and the program memory. The execute cycle of the jump does nothing but wait for the contents of the jump location to shift from the memory register into the instruction shift register. Then, with the next word strobe, the contents of the jump address are dumped into the instruction register. This timing is altered if the program memory and the data memory selected are the same since time for the data operations must be allowed within the program memory.

TABLE 8

INSTRUCTION TIMING

PM ≠ DM

PC	MA1	MD1	MA2	MD2	IR	SEQ.	OPERATION
0/-	-	R:0/-	-	-			
1/0	0	R:1/(0)	-	-			
2/1	1	R:2/(1)	-	R:A1/-	(0) = ADDM R	IND	(1) + (R) = A1 ⇒ MD2
3/2	2	R:3/(2)	A1	W:S/(A1)		EX	(A1) + ACC = S ⇒ ACC, MD1
4/3	3	R:4/(3)	A1		(2) = SKP ϕ	EX	ACC1 - ACC2: SIGN ⇒ TST
α/4	4	R:α/(4)	A1		(3) = JMP	IND(NOP)	(4) + (R) = α ⇒ MD1, PC
α+1/α	α	R:α+1/(α)	A1			EX(NOP)	
					(α) = NEXT		

PM = DM

PC	MA1	MD1	IR	SEQ	OPERATION
0/-	-	R:0/-			
1/0	0	R:1/(0)			
1	1	R:A1/(1)	(0) = ADD R	IND	(1) + (R) = A1 ⇒ MD1
2/1	A1	R:2/(A1)		EX	(A1) +(ACC)= S1 ⇒ ACC
3/2	2	R:3/(2)		NOP	
3	3	R:A2/(3)	(2) = ADDM R	IND	(3) + (R) = A2 ⇒ MD1
3	A2	W:S2/(A2)		EX	(A2) +(ACC)= S2 ⇒ ACC, MD1
4/3		R:4/S2		NOP	
5/4	4	R:5/(4)		NOP	
			(4) = NEXT		

All memory instructions except the jumps and literals require at least three memory cycles when the program memory and the data memory are the same, and for such instructions, the sequencer proceeds into its NOP states. If the instruction only calls for a data fetch from memory, three cycles are sufficient. However, if it calls for a store, then four cycles are required. The example shown in the bottom half of Table 8 illustrates two instructions, an ADD and an ADDM, using data from program memory. The add instruction is an example requiring only a data fetch. In this case the program counter is stalled during the index cycle, and during the execute cycle, the 11 read code normally hard-wired to the program memory bus is defeated to a 00 level so that the data memory bus ORing with it will introduce whatever codes are appropriate to the data operation. This will be a read code so there is no effective alteration. However, in the instruction following, the ADDM, the code transmitted on the data bus would be a write code. Since the next 2-bit memory cycle time must be devoted to writing, the earliest time at which the program counter contents can be restored as the memory address is in the following cycle. To account for this extra one-cycle delay, another NOP cycle is introduced and, additionally, the program counter is stalled during the execute state.

5.4.5 Instruction shift register.-- The instruction shift register uses the same typical 12-bit shift register as found in the rest of the machine. This register normally shifts in data from the output of the program memory switch. On one instruction, the modify instruction command (MDI), it switches to the output of the adder. The MDI instruction is in the class of those that cause an index operation although it is a single-cycle instruction. Thus, it treats the next instruction which follows as an address, performs an index on it, and shifts it into the instruction shift register. On the word strobe that follows, the modified instruction is dumped in parallel into the instruction register, where it is decoded and executed.

5.4.6 The program memory switch.-- The program memory switch is used to select the memory module which will be used as the source of the program for the logic unit. It performs a 16-way switching of two buses, one leading from the program counter to the memory and the other leading from the memory to the instruction shift register. It uses the standard 16-way switch configuration used in the data memory and register switching. A 4-bit register is set by the set program memory page (SPMP) instruction which transfers the low order four bits of that instruction into the register. The register is fully decoded for 16 states, and the decoders operate a pair of gates, one going to each memory from a common bus which is also the information entering the program counter, and the other from the memory to a common bus leading to the instruction shifting register. There are control buses to disable the switch for the condition of interrupt when the ZERO address must be sent to the program memory and when the returning data from the memory must be forced to ZERO.

The switch must also be disabled when the command unit forces an instruction into the instruction shift register. In this latter case the data from the command unit is ORed onto the bus leading to the instruction shift register. There are connections for two separate command modules. Each such connection includes a 4-bit decoder and data line. The codes which are used to address the command units overriding function are distributed throughout the system and each of the 4-bit decoders must be tied to the buses to reflect its own decoded address. The data bus from each command module is common. When the coded lines contain a correct address, the control level is brought up to disable the program switch, forcing its output to ZERO and the data from the command module is enabled to be ORed onto the bus to the instruction register.

5.4.7 The data memory and register select switching.-- The data memory switch is a 16-way switch identical to that of the program memory. It is controlled by the data memory register, a 4-bit register, the contents of which are loaded by the set data memory page (SDMP) instruction. This loads the four low order bits of that instruction into the data memory register. It is decoded to enable one of 16 two-way buses, coming from the adder output to the memory register and from the memory register to the B input of the adder.

Another similar switching section chooses the registers addressed in each instruction by decoding the six low order bits of the instruction. The register switches are of variable size in 16-bit increments. The first four bits of the instruction are decoded in common in all sections of the switch, while the high order two bits go through separate decoders and enable only one of the 16-bit sections.

The register address ZERO on the first switch section only is hardwired to the value ZERO to provide a register whose contents is always ZERO, and the first two register addresses of the same switch section connect to the accumulators. The register switch operates in conjunction with the memory switch, and both are ORed together, feeding the B input of the adder.

When IR7 is a ONE in the instruction register, it indicates a memory instruction, and the memory switch is enabled. The register switches are disabled by a signal called INHIBIT REGISTERS (INHR) which enters the decoders of the high order selection bits and disables them. When IR7 is a ZERO, signifying a register instruction, the memory switch is disabled and the register switch is enabled. An exception occurs when IR7 is a ONE during the index portion of that instruction. In this case, half of the memory switch is disabled, inhibiting the input from the adder, and the register switch is allowed on that bus. The other half of the memory switch controlling the adder input to the memory is enabled. Thus, the register selected feeds the input to the adder while the adder output feeds the memory. The register output is cycled back through the register switch to the register from which it came.

For instructions XCH, STA, NEG, STPC, SKDR, MSKR, and OUT, the adder output must be transferred to the registers, the loop from the register output back to the register input must be broken, and the register adder output must be inserted.

The input to the switch must also be changed for the timing signal transfer of bits 14 and 13, during which control codes must be transferred from the output of the adder to the register. For the index operation, the code generated is 11. The memory copies this directly, causing a read. It is inhibited during bit 14, however, on its way to the register, sending a 01 code to the register, or simply SHIFT.

5.4.8 Adder input switches.-- The two inputs to the adder are labeled A and B. Each one comes from a number of different sources. The B input is fed by the memory/register input bus except for those instructions coded with ZEROS in bits 7 and 8; in that case, the input from the memory/register bus is blocked to ZERO. For these instructions, the adder does not have to be a function of the data on this bus. Included are such instructions as XCHR and INP, where the data is significant but it enters the accumulators directly without going through the adder and instructions which use another input to the B port of the adder, the inverted memory/register bus. This inverted bus is used for the one's complement signal used in subtraction and is switched in by the subtract (SUB) and equivalence (EQVR) instructions as well as the NEGR. Another cause of forcing the input from the memory/register output bus to ZERO is during the multiply step when the value of the multiplier digit tested is ZERO.

Other inputs to the B port of the adder are the inverted outputs of the two accumulators. They are used only in certain skip instructions in which the accumulator entering the B port of the adder is subtracted from that entering the A port. The carry is set to ONE initially and the B input is inverted to accomplish this subtraction. Some of the skip instructions in this group require that the number being subtracted is ZERO. Since the carry is inserted as a general case, the ZERO is created by putting a third gate on the bus, which forces the bus to a constant ONE. This could be considered as a minus one and the addition of a carry to make it zero.

The A input to the adder also has a number of gates ORing into it. Primary among these are the two accumulators selected by the accumulator bit of the instruction (IR6). During the index cycle of the sequencer, the output of the program select switch goes into the adder, and the inputs from the accumulators are blocked. In one of the index instructions, JMPR, this gate is blocked to ZERO so that the index address from the register alone is the effective address of the jump. For the STPC instruction, the output of the program counter is brought into the A port of the adder. These then are the six data signals which are introduced into the A port of the adder: the output of either Accumulator 1 or Accumulator 2, the program counter, the program switch, forced to all ONES, or forced to all

ZEROS. The latter occurs in the NEGR instruction when the A port is forced to ZERO so that the contents of the register is subtracted from ZERO. In the skip and decrement (SKDR) instruction the whole bus is forced to all ONES so that the contents of the register are added to minus ONE.

5.4.9 The adder.-- The adder is a two-level min-term decoder of all the min-terms of 3 variables: the A input, the B input, and the carry, which will cause the sum to be a 1. Similarly, for the carry, the min-terms which will cause carry to be a ZERO are ORed. The two conditions of overflow are detected from thse gates and are strobed into the overflow flip-flop during the period the sign bit is being decoded. The two conditions are both sign bits ZERO and carry present or both sign bits ONE and no carry present. The carry flip-flop is set to ZERO with each word strobe and is enabled only by the instructions requiring an arithmetic result, such as ADD, SKP, MSTP, SUB, NEG, and SKDR. These instructions also enable the overflow detection. On the subtract and the negate instructions the carry is set to a ONE during bit 14 of the timing counter and this, together with the inversion of the input of the adder, provides the subtraction. The logic functions are obtained through the min-term decoders used for addition by forcing both C (carry) and $\overline{C}$ (not carry) to a ONE. This provides all four min-terms of A and B ORed into the adder, and the boolean functions AND and OR are obtained by selectively enabling the correct min-terms. The mask instruction is obtained with the addition of one more gate. The carry is a ZERO for this instruction, and the adder reflects an EXCLUSIVE OR of the two inputs A and B. One half of the EXCLUSIVE OR is disabled so that the function into the adder is $A\overline{B}$. If the accumulator is ZERO, B will be copied; otherwise, that gate produces ZERO. A separate gate AND's the two accumulators together, producing a ONE when the masking accumulator and the data accumulator both contain ONE. This is ORed into the adder on this mask instruction.

5.4.10 The accumulators.-- The logic unit has two accumulators which serve to provide one input to the adder and to receive its output. A single bit, IR6 in the instruction, determines which accumulator is to be used. The accumulators are made up of the typical 12-bit basic shift registers used throughout the machine connected as a LEFT/RIGHT shift register (see paragraph 5.2).

The data paths are selected by one of eight gates ORed together at the input of each accumulator. Two of these are for connection to the command module so that it may force data into the accumulators. The other six gates take data from various places within the logic unit. In general, the input is taken from the adder output. In the case of the exchange and load accumulator instructions, this gate is blocked and another gate is enabled, copying data from the memory/register output bus. This must be done since in the exchange instruction the adder is copying the output of the accumulator. This provides a path not through the adder but directly into the input of the accumulator. Having once done this for the exchange instruction, it is convenient to do it for the load accumulator instructions.

The gating of the skip (SKP) and shift (SHF) instructions must be treated as a pair since there are some shift instructions coded by using certain unused codes in the SKP instruction. For these, the gate copying the output is also disabled in addition to the gate copying the signal out to the registers. The gate's that are enabled by SKP and SHF are those that cycle the register upon itself and that extend the sign, or high order, bit leftward as the shift takes place.

For all instructions beginning with a 10 in the high order of the register field, the sign extended gate will be enabled. This includes the shift right arithmetic instruction (SRA) and long shift right arithmetic (LSRA). To create the shift right logical instruction, the cycle (CYC) (CYC) gate need only be disabled, thus shifting a ZERO into the left end of the register. The double length shifting instructions gate the same functions as already described to the input of Accumulator 1. Then, both the cycle and the sign-extended gate of Accumulator 2 are defeated and the output of Accumulator 1 is introduced.

The multiply step (MSTP) instruction OR's in with the long cycle instructions to enable the operative Accumulator 1 to feed Accumulator 2. At the same time, a separate gate disables the adder output from going to Accumulator 2. The output of the adder is read into Accumulator 1 and the output of Accumulator 1 shifts into Accumulator 2.

A special case for shifting right inputs is when the accumulators are addressed as registers. In this event, the adder output gate which would normally be enabled at this time is disabled and the bus which goes out to the registers is copied instead. A straight decoding of the R field register address levels accomplishes this. Both accumulators are located on the lowest 16-bit segment of the R field switch.

For shift left instructions, a decoding of the shift left instructions switches the accumulator to the shift left configuration. A ZERO is shifted into the lower order of Accumulator 2 under all conditions. The input instruction OR's with the shift left instruction to cause the register to shift left. A gate is enabled, introducing the memory/register output bus into the right end of Accumulator 1. On an input instruction, the data shifts in from the right.

Long shifts are coded by a 10 in the first two bits of the SKP instruction, differentiating it from the short shifts coded under the shift instruction. This 10 decoding is used to enable a path from the left end of Accumulator 2 into the right end of Accumulator 1 for the long shift left.

5.4.11 Accumulator clocking.-- With one exception, the accumulators shift on the shift clock pulses that occur during counts 12-1 of the timing counter. Counts 14 and 13 are excluded by a bus which gates them out of register transfers generally. This is a typical pattern throughout the machine.

The exception is when the multiply step instruction causes the accumulators to shift with the word strobe. Accumulator 1 is allowed to shift with the normal 12 SC pulses plus the word strobe, which is ORed in on the clock bus, and Accumulator 2 is allowed to shift with the word strobe only. Thus, Accumulator 1 performs a full 12-bit ADD and then, on a 13th pulse, the word strobe shifts out the low order bit of that sum into Accumulator 2, which also shifts once at that time on the word strobe.

The gating level controlling SC pulses comes from an OR of the control signal from the command module and a level from the shift and skip instructions, with bits 2 and 3 in the register field a 10. This latter class of instructions is the conditional skip or arithmetic comparisons between the two accumulators. Consequently, both accumulators must shift for these instructions. A third such ORed level comes from a decoder: the sequencer state 00 or execute, plus the accumulator's selection bit, which is ONE for Accumulator 2 and ZERO for Accumulator 1. In addition, the shifting of Accumulator 2 must be limited for certain instructions which are coded with a ZERO in bit 6 but which do not require shifting of accumulators. These are inhibited by two decoding circuits; the first decoding JMP and JMPR, the second decoding NOP, STPC, SMP, and NEGR. The only such instruction which unconditionally inhibits Accumulator 1 is skip and decrement (SKDR). The input (INP) and mask (MSK) instructions cause a conditional inhibiting when the mask bit coming from the low order bit of Accumulator 1 is a ZERO. The input and mask instructions also require the other accumulator to cycle to provide the mask, and, therefore, one of the ORed levels permitting shifting of Accumulator 2 comes from the decode of these instructions. The one remaining ORed signal that permits shifting of both Accumulator 1 and 2 decodes the double length shifts which require a shifting of both accumulators regardless of the accumulator bit.

Inhibiting the circuit which enables long shifts and the circuit which enables shifting on the proper accumulator bit is a circuit which is intended to halt shifting for shift instructions when the timing counter reaches ZERO. This circuit decodes the instructions skip (SKP) and shift (SHF) and a decode of ZERO from the timing counter. Actually, it is only the shift instructions which should cause the halt. Consequently, skip on bit set must be specifically excluded from this decoding by a separate circuit. The skip-on arithmetic comparison signal enters the logic in such a position that the shift halt circuit cannot be effected, thus not requiring this exclusion.

5.4.12 Timing counter.-- Timing of the logic unit operations is governed by a timing counter located in it. This is a 4-bit down counter which is initially set to the number 14 by the word strobe. It then counts down on each shift clock pulse to the number ZERO. During the counts of 14 and 13, the registered control codes are transmitted, and during the counts 12 through 1, the data transfers take place. In the case of shift instructions and certain skip instructions, the count is jumped after 13 to some number taken from the register field of the instruction. It then counts down that number of shift clock pulses and locks at the count of ZERO. In the case of shift instructions, this leads to shift N times, and

in the case of skip on bit N, the count of one strobes the bit of interest while the accumulators are allowed to shift a full 12 bits, as in other instructions. For most instructions the accumulators are allowed to shift for all counts other than 14 and 13 if the instruction uses them.

5.4.13 Skip.-- The skip instruction operates by performing a test during the execute cycle and inhibiting the copy next instruction (CNI) signal which would normally occur at the end of this cycle if the outcome of the test warrants. If the copy next instruction signal is inhibited at this point, the sequence counter will continue stepping through two no operation (NOP) cycles before again creating the copy next instruction signal. During this time, the next two locations in program memory will have been passed over and the third location following the skip will be copied as the next instruction.

There are two basic groups of tests for skip: skip on bit N and skip on arithmetic condition. Skip on bit N is coded with N from 1 to 12 in the register field. This number is used to jump the logic unit timing counter so that the count of 1 appears at the appropriate time to strobe the bit in question and set the test flip-flop.

The general form of the arithmetic skips is to defeat the signal jumping the timing counter so that the count-of-one strobe coincides as usual with bit 12 of the transfer. The opposite accumulator from that indicated by the accumulator bit in the instruction is then subtracted from the accumulator indicated, and the sign bit is used to set the test flip-flop. Since it must be possible to skip on the difference of any two numbers within the range of the machine, it is possible that the test subtraction may yield an overflow result. The output of the test flip-flop is EXCLUSIVE ORed with the overflow flip-flop so that the effect of the sign bit is reversed if overflow results from the subtraction.

A third condition of skipping is provided by overriding the strobe so that all bits of the difference are, in effect, strobed. With the test flip-flop locking on any ONE bit detected, this detects the condition that Accumulator 1 $\neq$ Accumulator 2.

These three skip conditions are doubled by forcing the accumulator indicated to be ZERO and tripled to a total of nine by forcing the number subtracted from it to be ZERO. The accumulator indicated is forced to be ZERO by blocking the input to the adder, and the number being subtracted from it is forced to ZERO by forcing the other input to the adder to be all ONES. Since subtraction is accomplished by adding the two's complement of the number being subtracted, the signal itself is inverted to form the one's complement, and an initial carry is introduced into the adder.

The one remaining class of skips is skip on overflow. The overflow conditions are detected by strobing the appropriate decoding circuits during the time when the sign bit is present. The two conditions decoded are: both the sign bit positive and a carry present, or both the sign bit negative and no carry present. These represent positive overflow and negative overflow respectively. The state of overflow flip-flop is set into the test flip-flop as the condition for skipping.

In addition to the 12 jump on bit set instructions, the nine arithmetic conditions, and the overflow, there are also instructions to skip on the complement of each of these conditions. Thus, there is also an instruction for skip on bit not set, a skip on no overflow, and such instructions as skip on Accumulator 1 < Accumulator 2, which is the complement of Accumulator 1 ≥ Accumulator 2. In total, there are two overflow skips, 24 bit skips, and 18 arithmetic skips.

5.4.14 Program counter.-- The program counter is made up of the same typical 12-bit shifting register used throughout the machine. At its input is a serial half adder with a carry flip-flop. This is used to serially add one to the program counter. The carry flip-flop is set to a ONE with each word strobe and continues to propagate as long as there are ONES emerging from the low order end of the register. This flip-flop is not initialized to ONE on the word strobe when the program memory switch is set to ZERO, preventing incrementing when the command override is in effect. That is, a ONE from the register and a ONE from the carry is the condition for propagating the carry into the next bit. In addition, there are gates for steering the input to the program counter under conditions of jumping and interrupting. A decoding of either the jump instructions plus the level indicating an indexing state in the sequence counter will force both halves of the half adder EXCLUSIVE OR to ZERO and enable a path from the adder output into the program counter. The jump instructions use an index cycle so that during the index cycle, the index result will be transferred from the output of the adder to the program counter. During the execute cycle of a jump, nothing takes place except waiting for the memory to react. There is also a gate forcing the output of the program counter to ZERO. This is used during interrupt, when it is desired to force the program counter and the memory address to ONE. Since the interrupt forced the previous contents of the program counter to ZERO, incrementing effectively forces the new contents to ONE. Details of this operation can be found in the discussion of the interrupt. There is also gating included to stall the program counter while data fetch cycles are taking place in the program memory under the condition that the program memory and data memory are the same. In this case, the shift clock to the program counter is blocked, preventing incrementing to the next location while the program memory is used for data.

5.4.15 The interrupt mechanism.-- Interrupt is accomplished by taking over certain control signals with a 2-bit sequencer called interrupt zero (INT0) and interrupt one (INT1). Exercise of this function is enabled or inhibited by a flip-flop called enable interrupt (EINT). Table 9 indicates the timing of the interrupt sequence. The slashes are used to indicate the contents shifting into a register on the left and the contents shifting out of it on the right of the slash. In this example the last instruction to be executed before interrupt is the contents of location 19 (LOC19). Upon occurrence of the interrupt signal, the INT1 flip-flop is set to a ONE. The decoding of this condition creates an inhibit signal which forces both the bus from the program counter to the memory and the bus from memory to the instruction register to ZERO. These two buses are not forced to ZERO until after the two control bits starting a

TABLE 9

INTERRUPT TIMING

PC	MD1		IR	R	INT1	INT0	EINT
20/19	20/(19)	(18)		-	0	0	1
21/20	0*/(20)	(19)	LAST INST	-	1	1	1
22/21	22/(0)		0*	-	1	1	0
1*/22	1*/(22)	(0)	STPC R	22	0	1	0
2/1	2/(1)		0*	22	0	0	0
-	-	(1)	INT ROUTINE	22	0	0	0
-	-		-	22	0	0	0
-	-		-	22	0	0	0
11/10	11/(10)		-	22	0	0	0
12/11	12/(11)	(10)	EINT	22	0	0	0
20/12	20/(12) = -2	(11)	JMP R	22	0	0	1
20/12	21/(20)			-	(1)	0	1
		(20)	NEXT INST				

*Forced by hardware

read cycle are sent to the memory. This is accomplished by gating this inhibit level with a timing level from the timing counter section of the logic unit. This timing level is high for the 12 data bit times only. This places a ZERO in the address of the memory and forces a ZERO or NOP instruction as the next instruction following the contents of location 19. The interrupt is gated with a copy next instruction (CNI) level so that it can only be initiated on the last step of the previous instruction. After one cycle, the contents of location ZERO then appear in the instruction register. This location should contain the store program counter register (STPC) instruction. This stores the program counter (which by this time is larger by 2 than the correct return address) in a register. The sequencer is then allowed to change the program counter by forcing the previous contents of the program counter, as seen in the incrementing half-adder, to ZERO. With the incrementing taking place, the address next created is ONE. Another ZERO must be forced in the instruction register to wait for the contents of location 1 to appear. Then, the contents of location 1 appear in the instruction register and the interrupt routine is begun.

In the course of sequencing through these control signals the enable interrupt flip-flop is cleared so that while in the interrupt routine, an interrupt cannot be received, as this would lose the original address. The last two instructions of the interrupt routine should be enable interrupt, (EINT) and jump (JMP) to location minus 2 indexed by R (which contains the return address plus 2), and this creates a correct effective return address. The ONE in parentheses indicates the earliest time at which a new interrupt could be received, since interrupt is gated by the copy next instruction (CNI) level and this occurs during the second cycle of a jump instruction.

5.5 I/O Register

The register control logic is shown in Figure 24, the state diagram in Figure 25 and the timing diagram in Figure 26. In this discussion the register mode connections are assumed, i.e., REG=1 and MEM=0. The 2-bit control code (CCD) transmitted by the logic unit is received in the control register (CR). The contents of this register will be described in the order CR2 to CR0 with X's indicating "don't care" digits. The CR is preloaded with the number 100 and as the control code shifts into it from the left, the 1 moves to the rightmost position: XX1. The first digit of the code (C0) determines whether data is to be loaded from the input channels into the shifting register or not. This transfer takes place during both input and output operations, although the data is used only for the inputting operation. In the output operation, the data shifting in replaces that loaded from the input channels, and the latter is never used. After the second code bit (C1) has been received, the register will shift on the remaining 12 SC pulses for all operations except no operation (NOP). The ONE in CRØ at this time activates the shifting gates rather than the parallel inputs. This ONE also inhibits further shifting of the CR by blocking the shift clock (SC). Thus, the control code remains in CR1 and CR2 until the WS pulse, which is not blocked, steps the register to its next state, which is 100 for all cases.

For the output operation, the WS pulse loads the buffer register with the data that has just been shifted into the shifting register. For the input or output operations, the WS pulse also sets the INPUT/WRITE or OUTPUT/READ flip-flops respectively. These are reset in every instance by the second SC pulse of the next operation which is gated by decoding X10 in the CR. These 2-microsecond pulses serve to signal the peripheral devices that data has been input or output respectively.

5.6 Memory Unit

The basic memory storage cell uses complementary MOS circuits for low standby power. The Memory Unit uses the I/O register module as both the memory address register and memory data register. Other LSI circuits are used for address decoding and bipolar-to-MOS interface circuits. Figure 27 is a logic diagram of this unit.

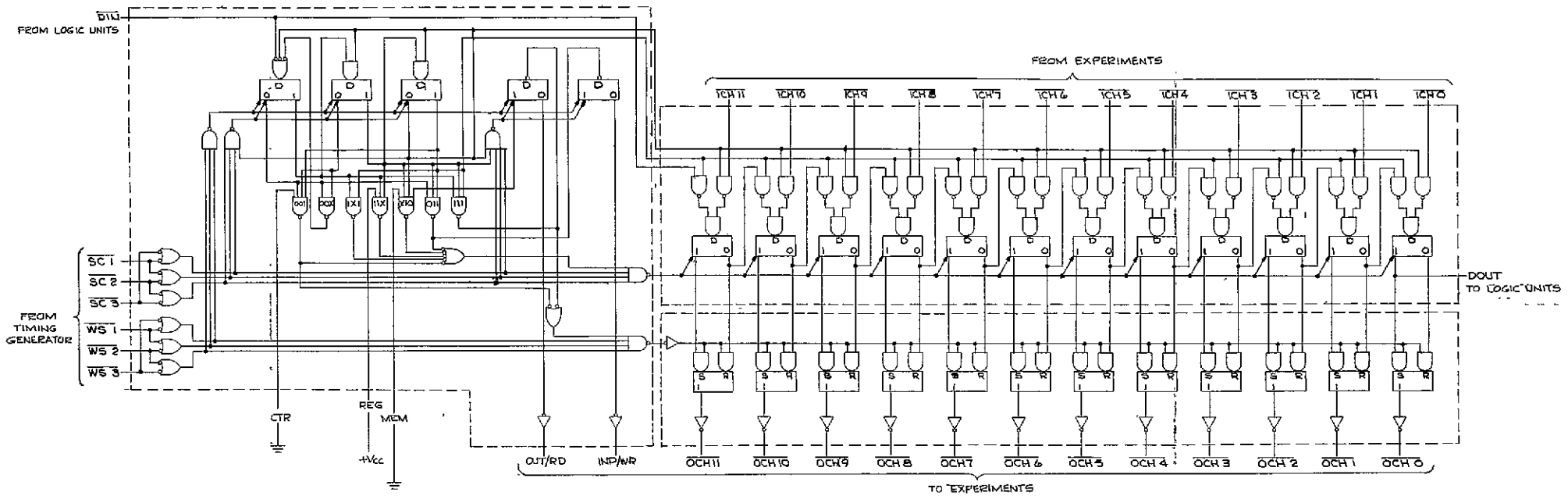
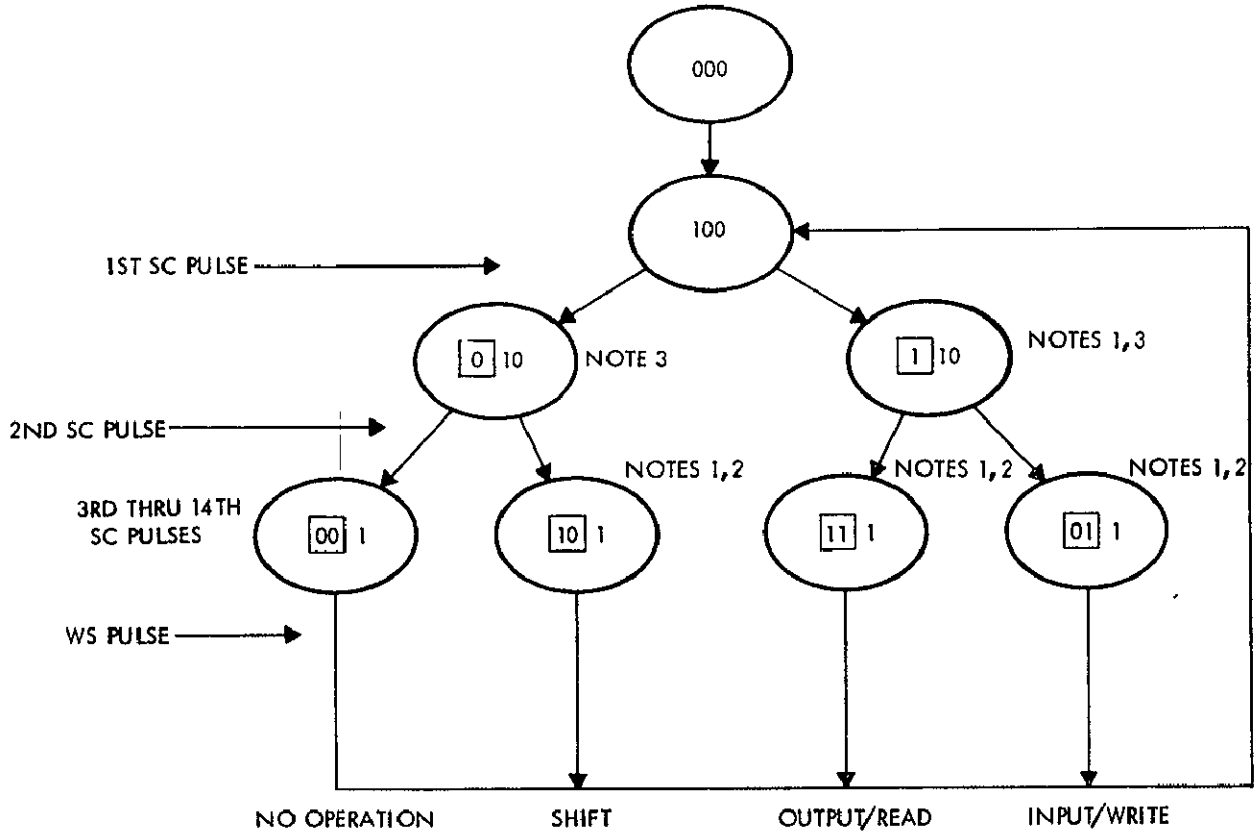


Figure 24. LSI MULTIPAC I/O Register

FOLDOUT FRAME

FOLDOUT FRAME 2



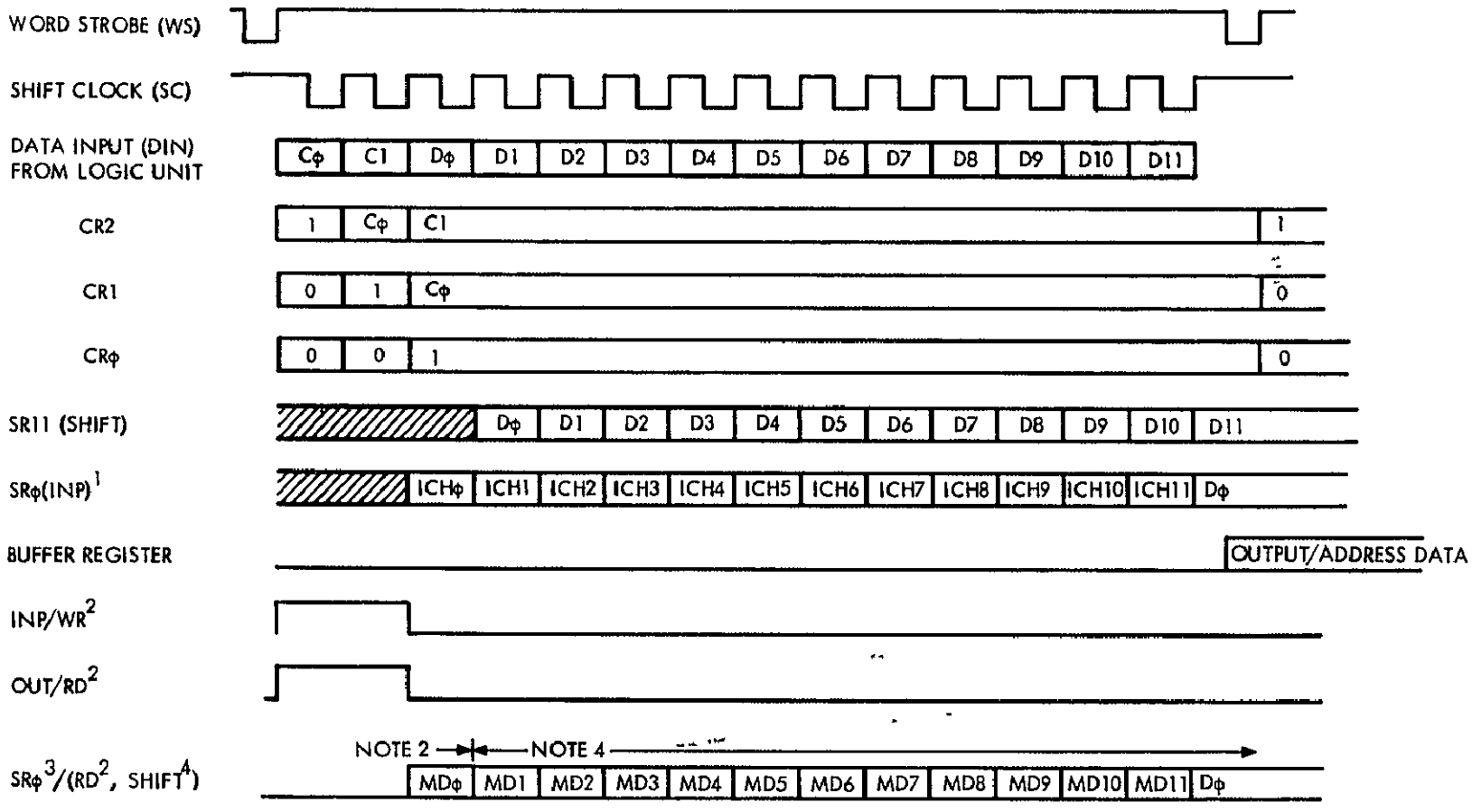
NOTE 1: DIGITS IN BOXES CONSTITUTE CONTROL CODES (CCD'S) AND ARE INPUT FROM THE DATA LINE INTO THE LEFT END OF THE THREE-BIT CONTROL REGISTER AS IT SHIFTS RIGHT.

NOTE 1: ENABLES CLOCK PULSES (SC) TO THE SHIFTING REGISTER IN REGISTER MODE.

NOTE 2: ENABLES CLOCK PULSES (SC) TO THE SHIFTING REGISTER IN MEMORY MODE.

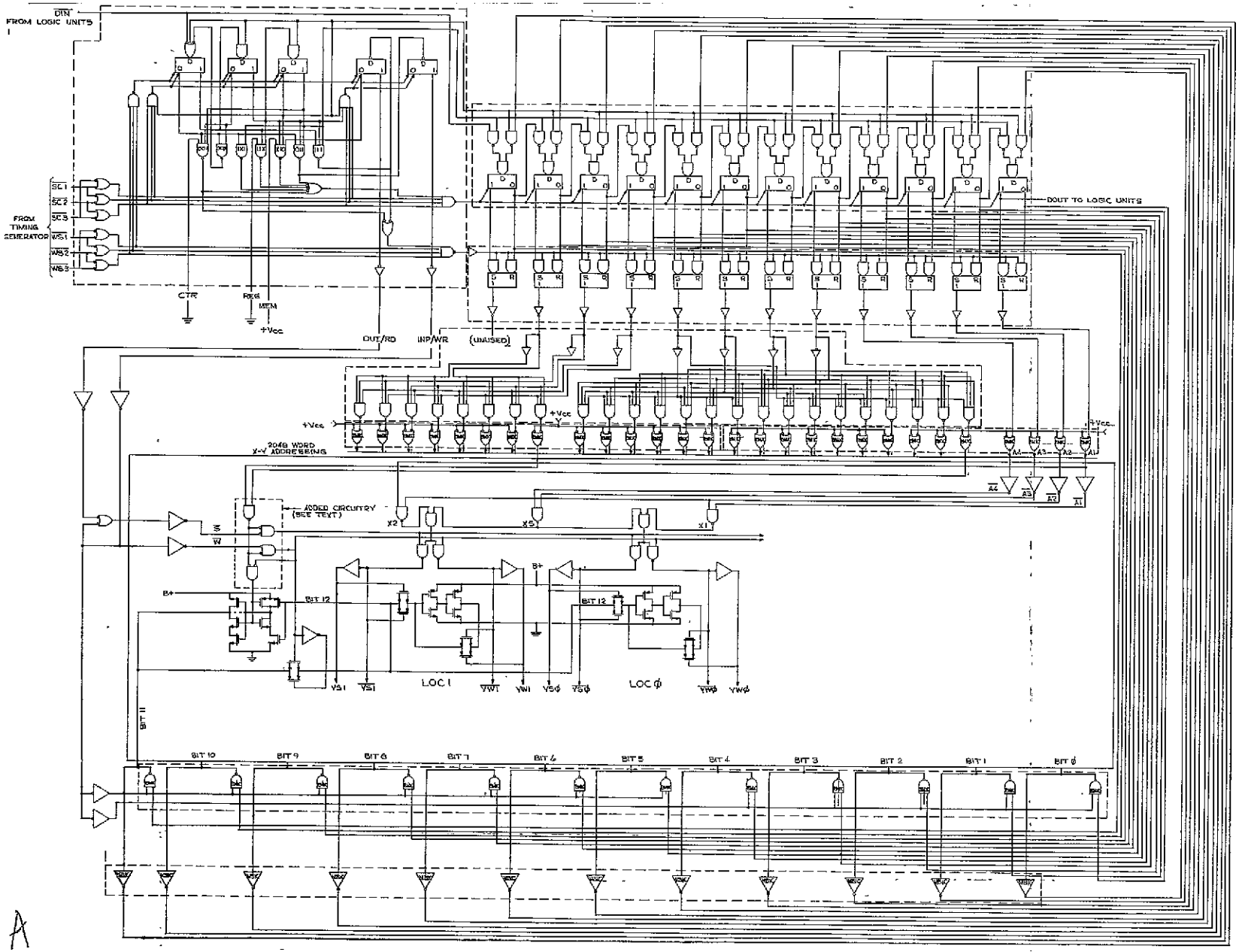
NOTE 3: ENABLES A CLOCK PULSE (SC) TO THE SHIFTING REGISTER IN MEMORY MODE IF THE THE INP/RD FF IS SET.

Figure 25. State Diagram of R/M Control Section



NOTE 1: REGISTER OPERATION ONLY
NOTE 2: GATED BY CONTROL CODE (CCD) OF PREVIOUS CYCLE
NOTE 3: MEMORY OPERATION ONLY
NOTE 4: GATED BY CONTROL CODE (CCD) OF CURRENT CYCLE

Figure 26. Timing Diagram of R/M Control Section



A

B

FOLDOUT FRAME 1

Figure 27. MULTIPAC Memory Unit (Typical Connections, Bit 11 Locations Ø and 1 Shown)

93/94

FOLDOUT FRAME 2

When the register portions which are the I/O register module are used in the memory selection, the address is transferred into the shifting register with a READ control code which causes it to be transferred to the address buffer. The OUTPUT/READ flip-flop is set for 2 microseconds following the WS pulse, and this serves the memory as a read strobe.

The MEM and REG input signals must be hard-wired to logic ONE and logic ZERO respectively for use of this circuitry in the memory. The changes which occur are indicated in Figures 25 and 26, the state diagram and timing diagram of the R/M Control Section. The change effected is basically that the loading of input signals into the shifting register is not enabled unconditionally by state 110 of the control register (CR) as it is in loading interface signals into the I/O Register. Instead, it is enabled by the state X10, which occurs in any event at the second SC pulse, and the ONE condition of the OUTPUT/READ flip-flop. Thus, after the read pulse has been up for 2 microseconds, the outputs of the sense amplifiers are loaded into the register.

Writing is accomplished by following the read operation with a transfer of data to be stored into the shifting register preceded by a WRITE control code. This causes the OUTPUT/WRITE flip-flop to be set by the WS pulse, and it remains up for two microseconds following the transfer. The memory address is not disturbed from the previous read operation; the data to be stored is read directly from the outputs of the shifting register since it is inactive during this 2-microsecond period. Thus, owing to the control logic used, the Memory Unit as a whole operates in a read-modify-write mode.

The modified register logic described above may be used to interface with any memory storage medium which can be accessed in the required 2-microsecond time. It is recommended that a 192-bit complementary MOS memory circuit be procured. This medium is the lowest powered circuit available, taking into account the drive circuit requirements of magnetic storage, and it has more than adequate speed. Sixteen and 32-bit chips are presently on the market and similar 64-, 256-, and 228-bit chips are under development by several manufacturers. The proposed memory organization shown in the block diagram has been derived from these sources and adapted to the MULTIPAC requirements, but it is strictly a preliminary design to guide preliminary estimates and contacts with possible suppliers.

Since bipolar circuits are used for system logic and complementary MOS for memory storage, interface circuits will be required. These would be typically 5-volt to 10-volt level converters for address and data inputs. The method of sensing the memory will govern the type of interface circuits from the memory to the logic, either voltage level converters or current sense amplifiers to approximately 5-volt logic levels as shown. Interface circuits, in general, must be carefully designed to minimize their power consumption since the relatively low impedances of bipolar circuits and high logic levels of MOS could lead to excessive power requirements. This problem can also be minimized by doing as much decoding on the memory chip as possible to minimize the number of address drivers.

The method chosen uses a two-way coincidence of a pair of 1 of 16 decoded levels to select a 16-word chip. The chip, in turn, decodes another 4 bits to select a single word. This requires only 36 level converters for a 4096-word memory. A 2048-word memory, shown in Figure 27, needs only 28 converters.

The storage device proposed is a modification of the 256-bit complementary MOS memory chip developed by Westinghouse under NASA contract number NAS-5-10243. The modification required consists of those measures necessary to permit use of the 16-word, 16-bit memory, which has been designed on a single chip, as a constituent part of a larger multiple-chip memory. Such expansion required a method of addressing a selected chip from among others, preferably a coordinate-select method, and a method of ORing the bit lines from each chip without extensive external circuitry. Either of these functions could be carried out in external circuitry but would require a larger number of extra gates and interface circuits to do so, whereas a slight modification of the chip would allow them to be accomplished by simple bussing interconnections of the memory chips themselves.

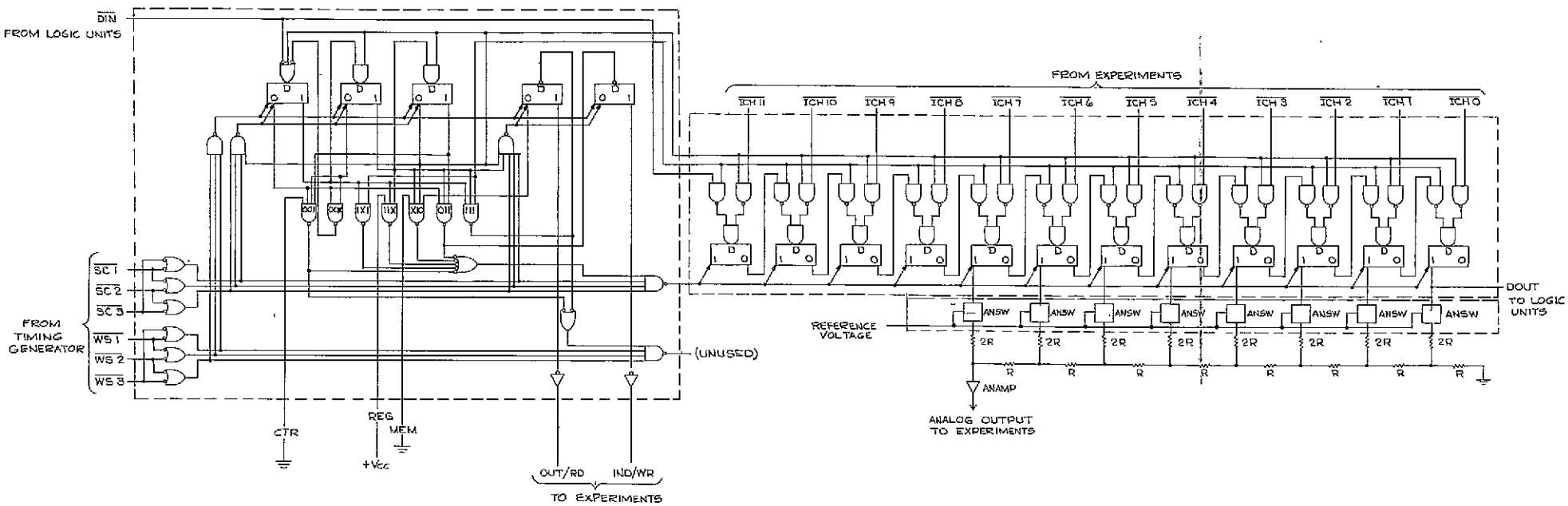
The logic diagram of the Memory Unit shows one possible such modification requiring only four gates and two additional external connections per chip. These modifications would permit X-Y selection of the chip in addition to the normal decoding of the word on the chip and wired OR operation of the bit lines. The latter would be high impedance from all but the selected chip during read, and high impedance from all chips during write. In this condition the write drivers are enabled to drive the bit lines.

The select signal ($\overline{S}$) of the memory is the OR of both READ and WRITE. This signal connects the internal flip-flops for the word selected to the bit lines. For the READ operation, these bit lines are gated to the output level converters (MBIC). For the WRITE operation, the internal cross-coupling of the flip-flops of the word selected are disconnected and the bit lines are driven to the input levels by the input level converters (BMIC).

The memory is volatile and hence will be destroyed upon power turnoff. If the memory is loaded prior to launch, then the spacecraft must be launched with power on the memory storage cells or the program will be destroyed. The power supply for the memory cells is separate from the logic since it is a different voltage. Since the standby power of the CMOS storage cells is extremely low, very little power will be drawn during launch.

5.7 D/A Register

The D/A Register (shown in Figure 28) uses two of the three LSI circuits of the I/O register: R/M Control and Shift Register. The buffer register circuit is replaced by a set of analog switches, a ladder network, and one or more isolation amplifiers. Only one isolation amplifier is shown, but more could be added in parallel if protection from one experiment shorting this output is desired. Some protection exists since there are two D/A Registers.



A

B

Figure 28. MULTIPAC D/A Register

FOLDOUT FRAME 1

FOLDOUT FRAME 2

The ladder network will most likely be discrete and the isolation amplifier will probably be a standard operational amplifier integrated circuit.

5.8 Command Unit

The Command Unit (shown in the logic diagram of Figure 29) accepts serial data from the Command Receiver in 16-bit words. It distinguishes between normal commands read by the program and overriding commands by the most significant four bits. A normal command has these four bits as all ZEROS. For these, it simply sets an I/O flag and holds the data until it is read by the program.

The other 15-bit combinations of these four bits specify one of three locations in one of five logic units. The three locations in each logic unit are its instruction register and its two accumulators. When such an overriding command is received, the Command Unit sends the 12 bits of data and the 4-bit address to all logic units, where each logic unit has hardware to decode the address and gate the data (if addressed) to the proper location.

In the allocation of register addresses, only two addresses are allocated for the two telemetry registers and the two command registers. If these registers are addressed by an input instruction, the command module is assumed; otherwise, the telemetry module is addressed. Thus, only one register control section is needed for a telemetry-command combination. No control section is shown on the Command Register logic diagram (Figure 29). It uses the INP/WR level from the control section of a telemetry register with the same address.

Data from the Command Receiver is shifted into the upper shift register of the Command Register with a clock supplied by the receiver. When all 16 bits are in, the receiver signals the register with a DATA FINISHED level. Two flip-flops are connected as a 2-bit shift register shifted with the word strobe to perform asynchronous-to-synchronous conversion of this level and obtain a one-time transfer level to gate the 16 bits into the other shift register and command address register portions of the Command Register. This level, obtained by decoding the first stage of the 2-bit register as a ONE and the second as a ZERO, will last for one period from one word strobe to the next. A set-reset flip-flop is then set to a ONE, and if this was a normal command (Command Address = 0), this signal which appears at the I/O interface on some input channel, may be sensed by the program. The INP instruction will set a control flip-flop on the second SC pulse (the first is counted out with a flip-flop) with the INP/WR signal from the R/M Control Section of the telemetry unit with the same address, and the following 12 pulses will shift the lower shift register. This control flip-flop and the set-reset flag flip-flop will be reset on the first SC pulse following the next word strobe. This clearing is performed by the same flip-flop which counted out the SC pulse previously because this flip-flop is cleared on every word strobe and is set to a ONE by every SC pulse.

When an overriding command (address ≠ 0) appears in the command address flip-flops, the same control flip-flop that INP set will be set and the lower shift register will be shifted by the same 12 SC pulses. Since the command address flip-flop levels are connected to all logic units, hardware at these units will decode these levels and open the correct gates, allowing the 12-bit data to be shifted into the proper register. At the next strobe, the command address register will be cleared in addition to the control flip-flop.

The input connections from the Command Receiver also are wired to an external input connector to facilitate loading of programs on the ground before launch, when the Command Receiver is off.

5.9 Telemetry Unit

The Telemetry Unit interfaces with the modulator of the telemetry transmitter which is used to transmit the spacecraft data to the ground station. The design of this unit is highly dependent on the final design of the telemetry modulator. (It is assumed here that the convolutional encoder is part of the telemetry modulator and any switch to bypass the encoder is in the modulator also.) The design presented here assumes that the modulator accepts data serially one bit at a time. If, instead, the modulator would accept 12 bits at once, then these need no special telemetry register. A standard I/O Register would suffice. This pushes the unique logic in the present telemetry register into the modulator which, of necessity, has to be specially designed and fewer module types would be needed for the MULTIPAC system.

The detailed logic diagram of this unit is shown in Figure 30. Two of its circuits are the same as an I/O Register. The other two circuits are the Telemetry Buffer Register and associated logic. The buffer register shifts, including a 1-bit high order extension of it, on the telemetry clock pulses. A 1-bit extension flip-flop is preset to a ONE when data is transferred to the buffer register. Shifting fills the buffer register from the left with ZEROS. When the ONE reaches the next-to-low-order stage, the state 0002_8 or 0003_8 is decoded by a 12-input NAND gate. This gate causes the next telemetry clock pulse to load a new telemetry data word in parallel from the shift register and to preset the extension bit again. When the extension bit is set, a set-reset flip-flop (shown as cross-coupled NAND gates) is set to serve as a flag to the processor to advise it that the next telemetry word should be transferred into the shifting register. On an output (OUT) instruction, the flag flip-flop is cleared by the word strobe to prevent ambiguity.

The telemetry clock, in the non-coherent telemetry mode, will most likely be divided down from the basic oscillator in the Timing Generator, not the word strobe. Thus, in both this case and in the coherent mode, the telemetry clock will be asynchronous with the telemetry clock signals. The flag flip-flop must be set synchronous with the word strobe so that it cannot be set during or after reading the flag and yet be automatically

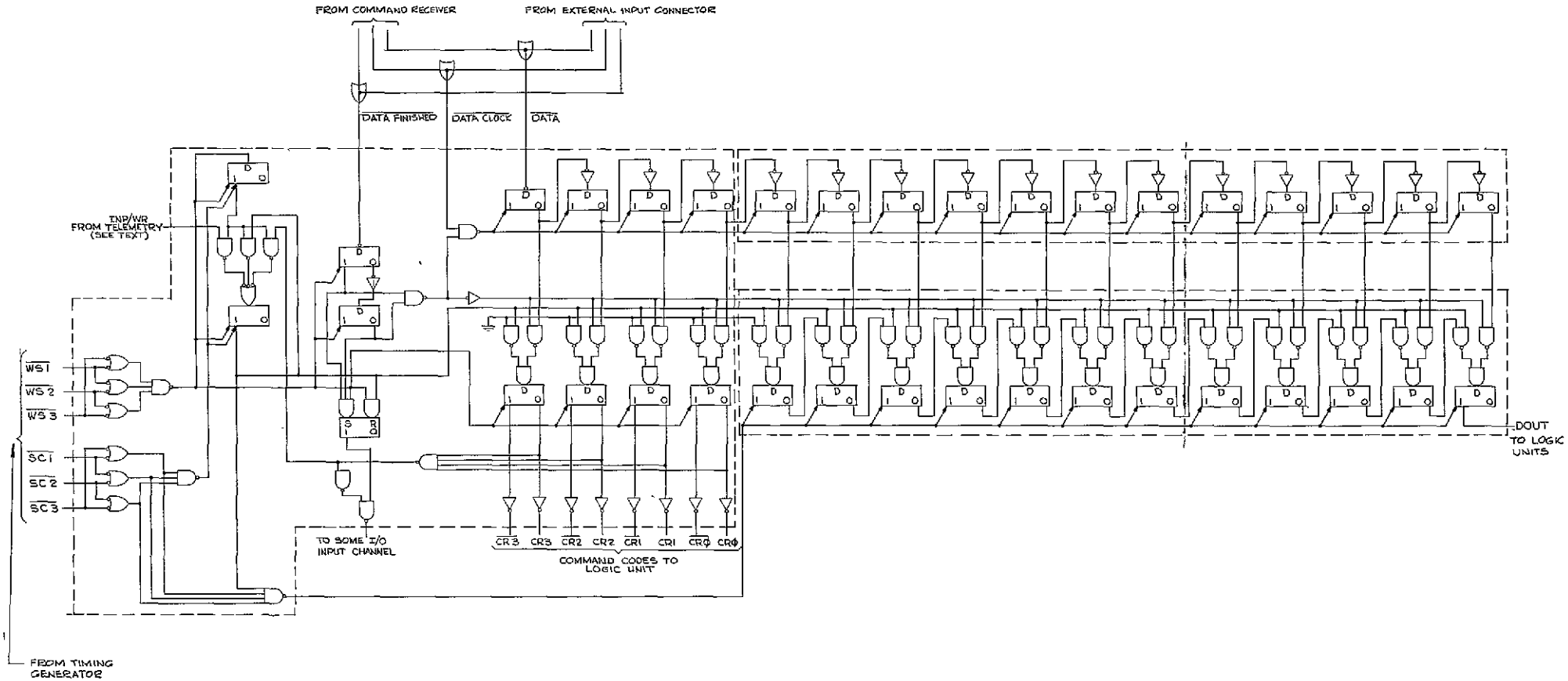


Figure 29. MULTIPAC Command Register

A

FOLDOUT FRAME

B

101/102

FOLDOUT FRAME

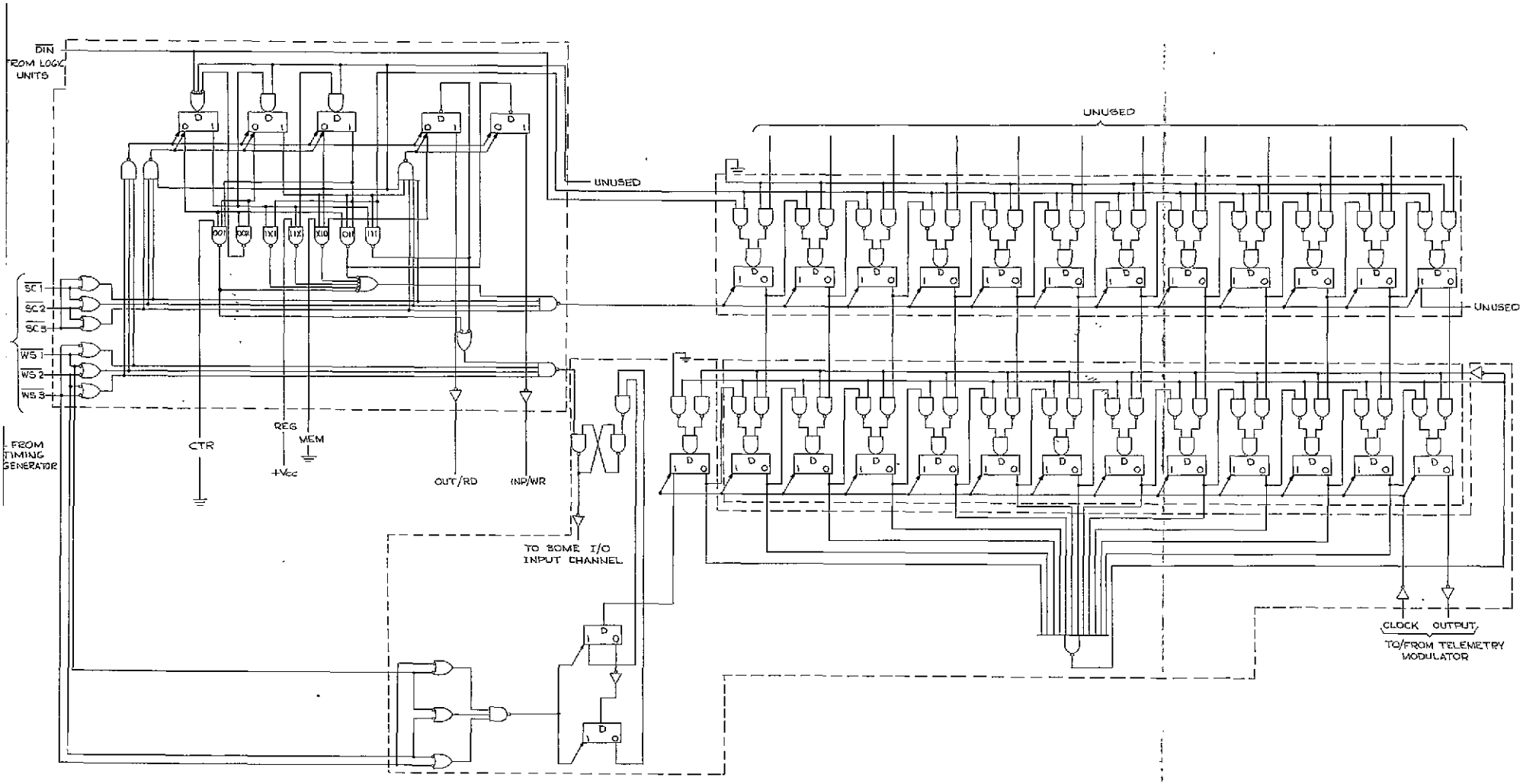


Figure 30. MULTIPAC Telemetry Unit

A

FOLDOUT FRAME 1

B

FOLDOUT FRAME 2

cleared on an OUT instruction. Two flip-flops are used to convert the asynchronous occurrence of need for more data in a manner identical to synchronizing the DATA FINISHED signal in the Command Unit. The setting of the extension bit to a 1 is used for the data request information since this guarantees that the previous data is transferred to the lower register. The word strobe majority logic is duplicated since those signals are not available outside the R/M Control LSI circuit.

The input interface signals on this module cannot be used since INP instructions refer to the Command Unit.

Some logic in the control section and input gating to the unused input channels could probably be eliminated, but in the interest of reducing LSI chip types, it is identical to the I/O Register. As in the I/O Register, REG and MEM must be wired to ONE and ZERO respectively.

5.10 Timing Generator

The Timing Generator, shown in the logic diagram of Figure 31, provides both the shifting clock (SC) pulses and the word strobe (WS) pulses to all other modules. Each of these signals is supplied in triplicate throughout the system and is decoded by majority voting gates at each module interface. The Timing Generator is driven by 1-MHz square waves. One of two 1-MHz oscillator and squaring circuit combinations is selected by the Command Receiver which will switch from one to the other with a special command.

Each of the three counters is a feedback shift register. The register shifts right each clock pulse, with the input to the first stage equal to the EXCLUSIVE OR of the two rightmost stages. This counter will sequence through the states shown in Table 10. The state 0000 (all stages equal to zero) is not allowable for this type of feedback. In this case, the register would normally "hang up" and continually stay in the ALL ZEROS state. This condition is used to put all counters in synchronism. When the left three stages of the shift register are ZEROS (states 0000 or 0001), the feedback is disabled and the counter will not continue until at least one other counter is in the same state. When this counter and one of the others is at 0000 or 0001, a ONE will be fed into the leftmost stage of all three counters. At the same time ZEROS are forced into the other three stages to guard against the possibility of failure of one counter permanently in the 0000 or 0001 state.

In this manner a counter will hang up until one other is in synchronism, and as soon as two are in synchronism, all three are forced to the 1000 state. If one counter fails, the other two will maintain synchronism, assuming that two of the three SC and WC pulses will be correct.

TABLE 10

STATES OF THE TIMING COUNTER

0	0	0	1
1	0	0	0
0	1	0	0
0	0	1	0
1	0	0	1
1	1	0	0
0	1	1	0
1	0	1	1
0	1	0	1
1	0	1	0
1	1	0	1
1	1	1	0
1	1	1	1
0	1	1	1
0	0	1	1

5.11 Real-Time Counter

The Real-Time Counter is made up of the three circuit types of the I/O Register and one additional special circuit entitled Increment and Control. This new LSI circuit, shown in Figure 32, will allow the shift register to be incremented every word-time in an identical manner to that of the program counter of the logic unit. If the input labelled COUNT is at zero, then the flip-flop controlling the incrementing will be set to zero at the word strobe and therefore no counting will take place. This feature is used to expand the Real-Time Counter beyond 12 bits. To expand, we take the output from the incrementing control flip-flop (labelled OVERFLOW on Figure 11) and tie this signal into the COUNT signal on another Increment and Control circuit. This overflow signal will be a ONE when all 12 bits of the register are ONES and the word is being incremented to all ZEROS. This is the only time that this flip-flop will be a ONE at word strobe time, which will set a ONE into the incrementing control flip-flop, allowing the shift register tied to this Increment and Control circuit to increment once every time the preceding 12 bits overflows. The low order 12 bits of the Real-Time Counter will have the COUNT signal tied to plus V_{cc} so that it will count at every word-time.

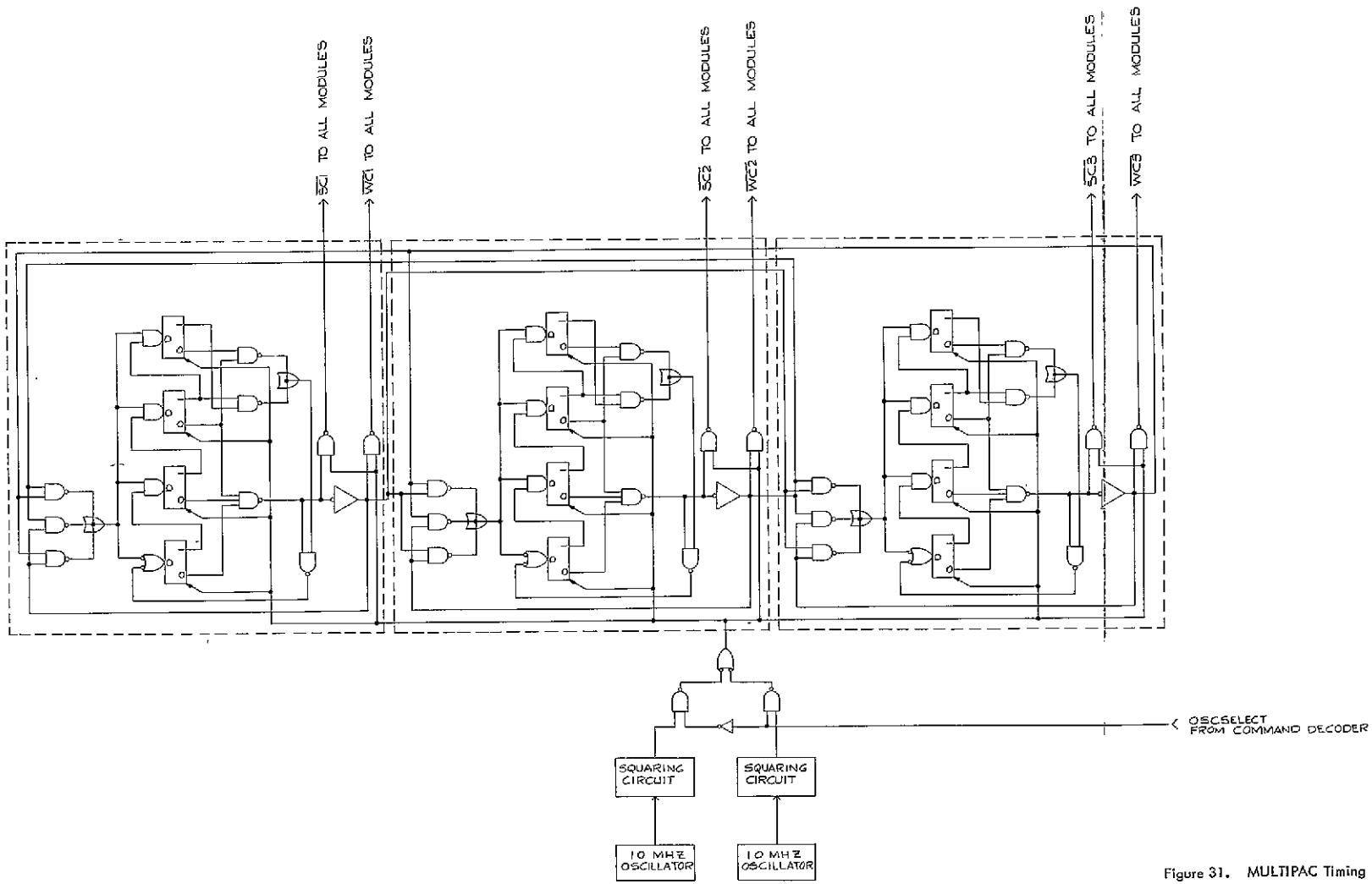


Figure 31. MULTIPAC Timing Generator

A

FOLDOUT FRAME 1

B

FOLDOUT FRAME 2

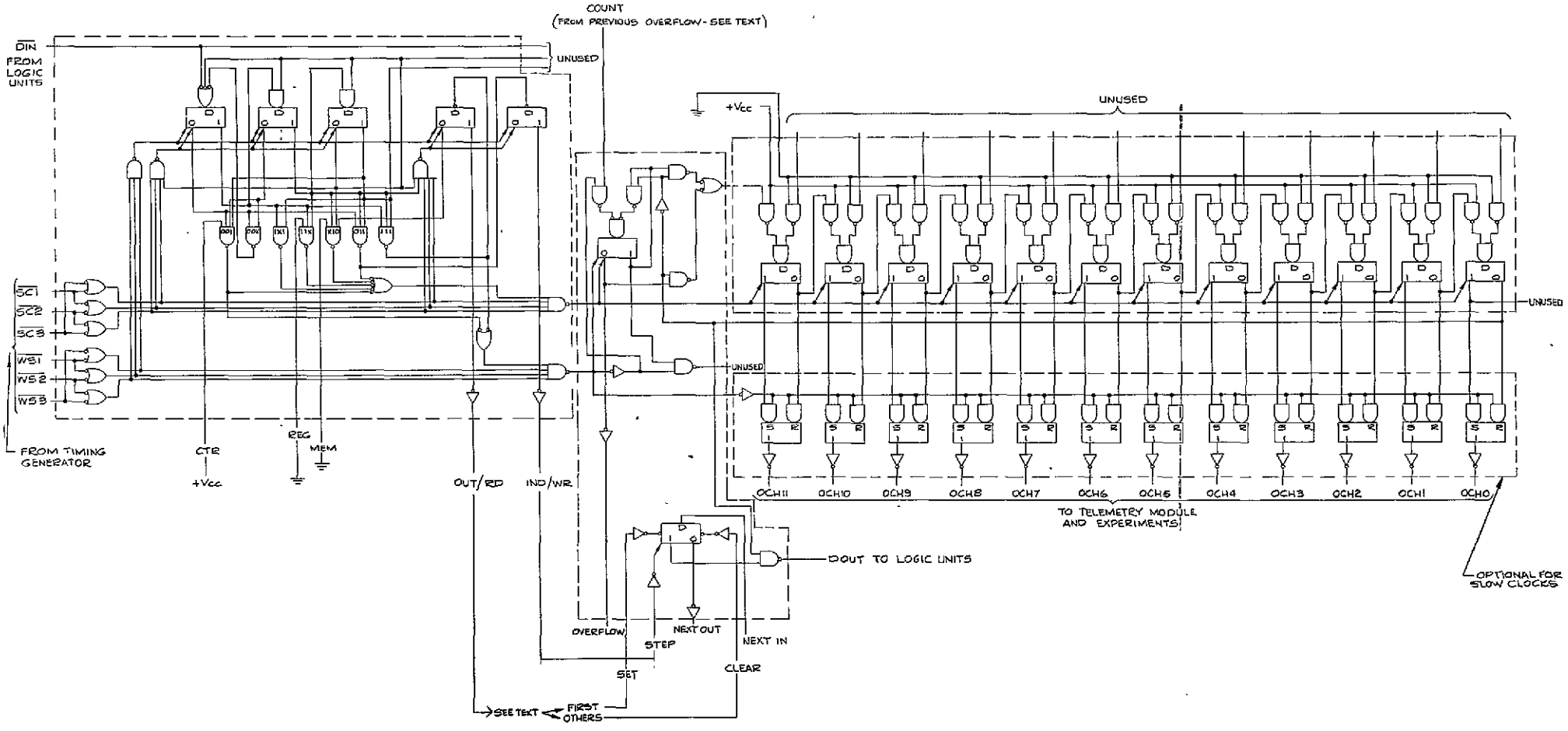


Figure 32. Real-Time Counter

A

FOLDOUT FRAME 1

B

FOLDOUT FRAME 2

In Figure 11 an output buffer register is shown tied to the shift register. This register is not needed for operation as a Real-Time Counter, but its addition allows the generation of slow clock waveforms which are binary multiples of the basic word-time. This buffer register will be loaded once every word-time with the present state of the shift register and will have the effect of a slowly counting register with a parallel count output, instead of the continuous shifting of the shift register. These slow clocks may be useful for some experiments and also for the telemetry system when in noncoherent mode. These buffers need only be added to those 12-bit sections for which slow outputs are needed. This will probably be only the low 12 bits of the Real-Time Counter.

When the Real-Time Counter is longer than 12 bits, which for most applications will probably be 36 bits, a method of choosing which section to read for an input instruction is desirable to save using MULTIPAC register addressing.

Logic is included in the Increment and Control circuit to reduce the number of MULTIPAC register addresses needed for Real-Time Counters longer than 12 bits. For most applications, 36 bits will be used, giving a time scale of slightly less than 2 weeks. Rather than use three addresses to input the three 12-bit words representing the Real-Time Counter, the hardware in the Increment and Control circuit will sequence through each 12-bit section each time an input instruction addresses the Real-Time Counter. An output instruction will reset this sequencing logic so that the next input instruction will address the least significant 12 bits, the second instruction will address the second least significant 12 bits, and so forth. This is accomplished by connecting a special flip-flop in each Increment and Control circuit together as a ring counter. An output instruction will set a ONE in the flip-flop of the least significant stage and ZEROS in all other stages. Each input instruction will read that 12-bit shift register for which the flip-flop is set to a ONE and shift the ONE to the flip-flop in the next Increment and Control circuit.

5.12 Sample Rate Counter

The Sample Rate Counter (Figure 33) uses the Register Control circuit and the shift register circuit of the I/O Register and the Increment and Control circuit of the Real-Time Counter. This circuit is similar to the Real-Time Counter in that if the count input is a ONE at word strobe time, the shift register will be incremented. Like the Real-Time Counter, the least significant 12 bits will always count every word-time because the count will be wired to $+V_{cc}$ and the other 12 bits sections will be controlled by the overflow of the previous 12 bits. For many applications only one section of 12 bits will be used.

The major difference between the Sample Rate Counter and the Real-Time Counter is that the Input signal to the shift register section, instead of being unused as in the Real-Time Counter, is tied to an output buffer of some I/O Register. When the overflow signal is present at word strobe time (which only occurs when all 12 bits are one), the word strobe will be ORed with the clock pulse of the shift register. At this time the

input signal will be high and shift level will be low and the 12 bits from the output buffer will be loaded into the shift register. Thus, after all 12 bits have overflowed, the count will start at a number determined from the bits stored in an output buffer from an I/O Register, allowing control of the sample rate by the program.

When the counter overflows, the overflow signal is wired to the other flip-flop in the Increment and Control circuit to be used as an interrupt signal to one of the logic units. This flip-flop will be cleared by an output instruction to this counter. Thus, the program must send an output instruction to the Real-Time Counter after receipt of an interrupt.

In addition to the word strobe time following an overflow, the output buffer containing the reset number will be inputted to the sample rate counter on receipt of an input instruction in a manner similar to a normal I/O Register. This permits the program to resynchronize the sample rate counter.

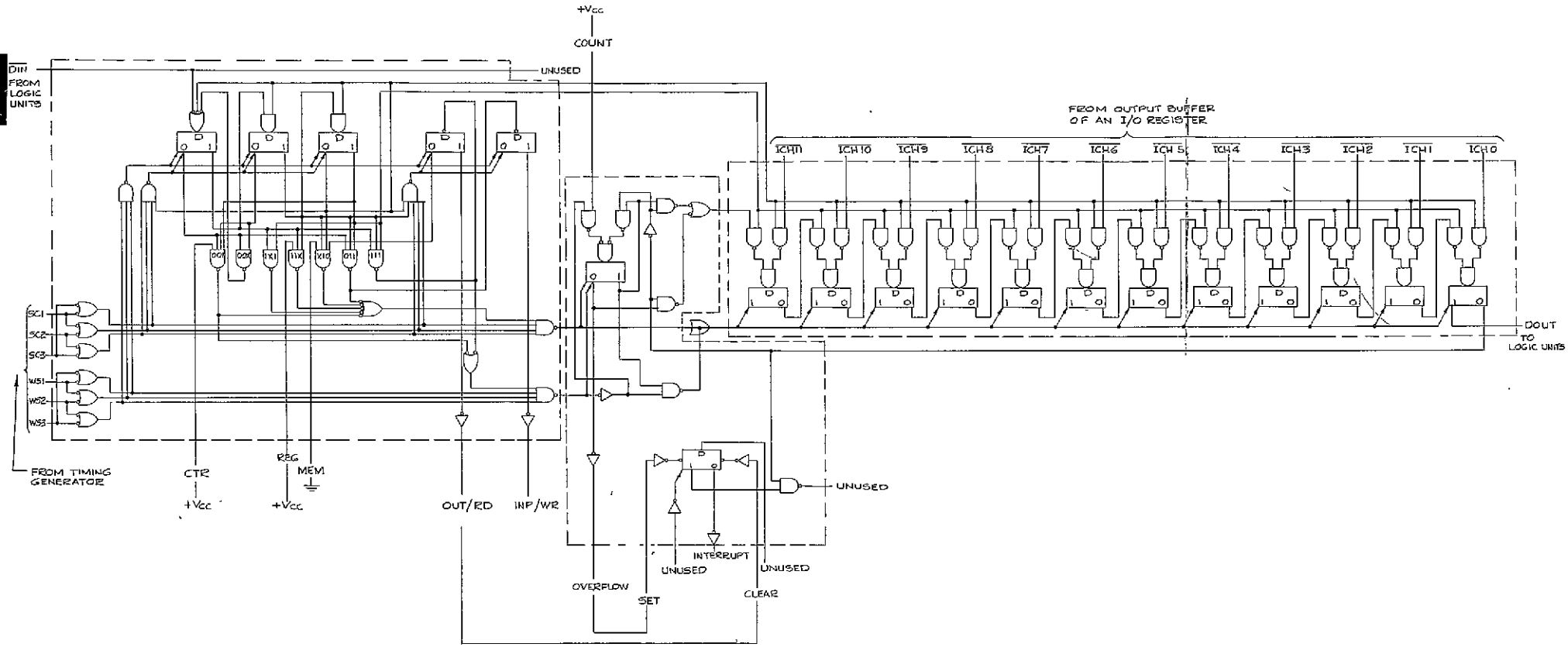


Figure 33. Sample Rate Counter

A

B

FOLD ... FRAME

FOLDOUT FRAME

6.0 RELIABILITY

The reliability of the MULTIPAC system depends largely on its restructurability in the event of failure. Since the initial configuration makes use of all modules in the system, such restructuring to delete failed modules leads to a progressively simpler machine and consequently to a gradual degradation of the processing capability. Thus, to state the reliability of the system, or the probability that it will be operational after some specified period of time, one has to define the configuration under consideration. The initial configuration has a very low probability of lasting for the duration of a lengthy mission without requiring some restructuring for repair purposes. There is a high probability, on the other hand, that the minimal processing mode can still be implemented from the surviving modules after years of system operation. Intermediate modes of operation has probabilities of survival between these extremes according to the module requirements, and hence, the processing capacities of such modes.

The organization of MULTIPAC permits two basic methods of restoring operation by reprogramming after failures occur. In the first, extra modules not needed for the initial configuration may be included to be used as spares should failures occur. In the second, the remaining modules may be reassigned in a different configuration which will continue to take care of the highest priority tasks. Such a loss in capability can best be compensated for by accepting a reduction in the telemetry rate or discontinuing some on-board processing tasks. This may be continued through a number of failures and successively less powerful configurations until the minimum operational configuration is reached.

Reliability thus becomes a function of both the modules initially available in the system flown and the configuration for which the probability of survival is specified. Figure 34 gives the reliability model for the typical system in its initial configuration showing only actual spares replacing one another.

In addition to the number of processors which can be configured, a gradual degradation of the multiplexed connections to the experiments is also to be expected, and this is reflected in the probability of a certain percentage of the I/O lines remaining available in each of these modes. In Figure 34 a typical connection of the I/O interface is assumed for modelling purposes. Twelve of the 25 available I/O registers are connected to 72 digital signals, each appearing redundantly at the inputs of two different I/O registers, for example R_1 and R_7. Another 72 are used for analog conversion signals and are connected to the outputs of 72 comparators with the same redundancy, for example R_{13} and R_{19} are redundant. The comparators connected to input channels on R_{13} through R_{18} are driven by the reference signal from D/A 1 while those connected to R_{19} through R_{24} receive the reference from D/A 2. Special modules such as a sample rate counter and a real-time counter are not related in this model.

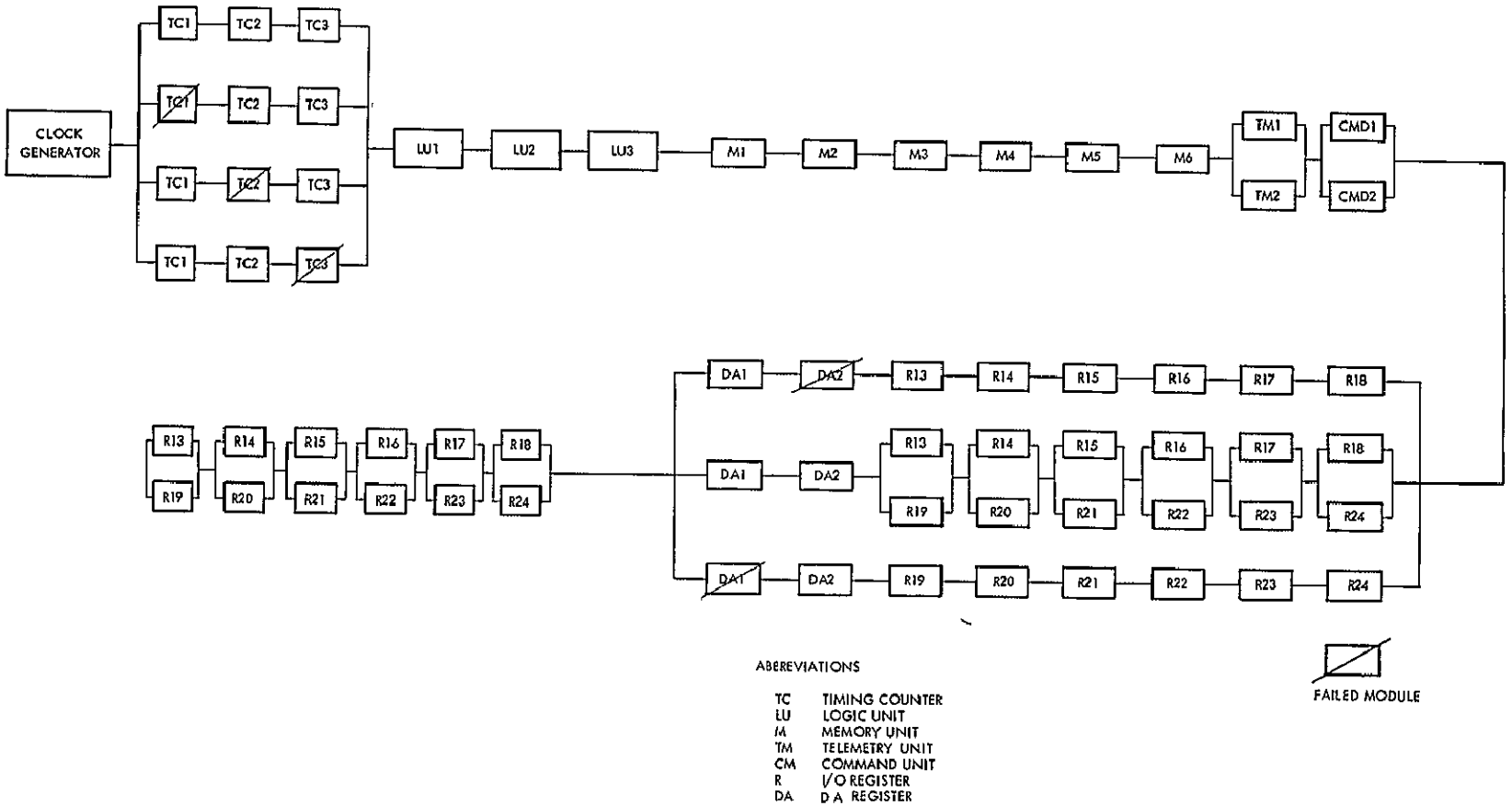


Figure 34. Reliability Model of LSI MULTIPAC

A computer program was written to calculate reliability figures for M of N modules which shows the higher probabilities of survival attainable for degraded operation. The series reliability given for four such modes of operation is shown in Table 11. The program and its output is included as Appendix A. Table 11 gives these figures for 12, 24, and 36 months and for failure rates of 10^{-7}, 10^{-6}, and 10^{-5} per LSIC hour. It is expected that circuits can be produced which will have a failure rate somewhare between 10^{-7} and 10^{-6}. A failure rate much lower than 10^{-6} will not produce a reliable system as can be seen from the results of a 10^{-5} failure rate in Table 11. Each of the module-failure rates (except memory) reflects the conservatism that a single failure disables the entire module. Table 11 ignored the memory storage elements on the assumption that if a few failed, enough memory remained to continue operating with negligible reduced performance. Table 12 includes the memory storage elements and assumes that the failure of one causes the entire memory to fail. The actual reliability figures will be somewhere between these two extremes.

The memory storage elements represent 16 consecutive locations in memory. A single failed storage element will be easy to program around and is a very small percentage of storage. A large number failed at random locations in one memory unit may be difficult to program and, in fact, make the memory useless. It can be shown that there is a high probability of most of the cells surviving: 0.9995 for 118 of 128 cells surviving 36 months at a failure rate of 10^{-6}. This would destroy 160 of the 2048 storage locations and, given the proposed addressing scheme, they would, at worst, consist of 10 separate blocks of sixteen words each so long as the failures did not propagate on the data sense or addressing lines. For the latter type of failures, some special definition of their probability is required from the manufacturer. In Table 11 the reliability of the storage cells is considered to approach one for a sufficient number of them surviving to make the module usable, and thus only the control and addressing logic is considered in the reliability calculations.

The combination of the two oscillators, two squaring circuits, and the switch was assumed to have a failure rate of 0.2 of an LSIC.

Included in Tables 11 and 12 (same in both) are the probability of a given experimental line (digital or analog) surviving if the remainder of the system is operable. This is looking at it from the experimenter's viewpoint, instead of at the mission as a whole.

The formulas used in the calculations are as follows:

$$MR = e^{-(PC)(FR)(730)(MO)}$$

and

$$COMBR = \sum_{NS=NMR}^{N} \frac{N!}{NS!(N-NS!)}(MR)^{NS}(1-Mr)^{N-NS}$$

TABLE 11

LSI MULTIPAC SYSTEM RELIABILITY

Failure Rate = 10^{-7} LSIC's per LSIC-hour

Mission Duration =	12 months	24 months	36 months
3LU's, 6 M's, FULL I/O	.9238	.8531	.7874
2LU's, 4M's, FULL I/O	.9992	.9970	.9935
1LU, 2M, FULL I/O	.9996	.9987	.9974
1LU, 1M, 83% I/O	.9998	.9995	.9991
Probability of being able to communicate with any experiment			
Analog	.9998	.9994	.9986
Digital	.99999	.99995	.99987

Failure Rate = 10^{-6} LSIC's per LSIC-hour

Mission Duration =	12 months	24 months	36 months
3LU's, 6M's, FULL I/O	.4444	.1905	.0793
2LU's, 4M's, FULL I/O	.9366	.7836	.5973
1LU, 2M, FULL I/O	.9761	.9124	.8191
1LU, 1M, 83% I/O	.9928	.9692	.9254
Probability of being able to communicate with any experiment			
Analog	.9854	.9477	.8948
Digital	.9982	.9910	.9770

TABLE 11 -- continued

Failure Rate = 10^{-5} LSIC's per LSIC-hour

Mission Duration =	12 months	24 months	36 months
3LU's, 6M's, FULL I/O	.0001	.0000	.0000
2LU's, 4M's, FULL I/O	.0185	.0000	.0000
1LU, 2M, FULL I/O	.1522	.0021	.0000
1LU, 1M, 83% I/O	.3307	.0129	.0001
Probability of being able to communicate with any experiment			
Analog	.4430	.1035	.0187
Digital	.7190	.2910	.0849

TABLE 12

LSI MULTIPAC SYSTEM RELIABILITY WITH FULL MEMORY STORAGE

Failure Rate = 10^{-7} LSIC's per LSIC-hour

Mission Duration =	12 months	24 months	36 months
3LU's, 6M.s, FULL I/O	.4727	.2233	.1055
2LU's, 4M's, FULL I/O	.9774	.8834	.7390
1LU, 2M, FULL I/O	.9995	.9966	.9865
1LU, 1M, 83% I/O	.9998	.9994	.9984
Probability of being able to communicate with any experiment			
Analog	.9998	.9994	.9986
Duration	.99999	.99995	.99987

Failure Rate = 10^{-6} LSIC's per LSIC-hour

Mission Duration =	12 months	24 months	36 months
3LU's, 6M's, FULL I/O	.0005	$.2 \times 10^{-6}$	$.1 \times 10^{-9}$
2LU's, 4M's, FULL I/O	.0696	.0008	$.6 \times 10^{-5}$
1LU, 2M, FULL I/O	.5741	.0909	.0090
1LU, 1M, 83% I/O	.8800	.4280	.1461
Probability of being able to communicate with any experiment			
Analog	.9859	.9477	.8948
Digital	.9981	.9910	.9770

where

MR = reliability of a single module

PC = parts count of LSIC's

FR = failure rate per LSIC-hour

730 = hours per month

MO = mission duration in months

N = number of modules in system

NS = number of modules surviving

MNR = minimum number of modules required

COMBR = combinational reliability of at least MNR of N modules surviving

The formula for combinational reliability is diagrammed for 2 or 3 timing counters in Figure 32. The same approach is used, though not diagrammed, for all other MNR of N cases. The three branches of the DA section describe three mutually exclusive sample spaces in which some MNR of N registers constitute a further requirement for success. This shows the method of analysis for the probability of at least a certain portion of the I/O surviving. The 83 percent figure given is based on losing one of the six register pairs providing entirely digital I/O and one of the six register pairs providing the analog inputs.

MULTIPAC INSTRUCTION LIST
(Per order of appearance in Section 7.0)

Mnemonic	Operation Code	Operation	Page
LDA1	76XX	Load Accumulator 1 from Memory	127
LDA2	77XX	Load Accumulator 2 from Memory	127
LDA1R	74XX	Load Accumulator 1 from Register	128
LDA2R	75XX	Load Accumulator 2 from Register	128
STA1	52XX	Store Accumulator 1 in Memory	129
STA2	53XX	Store Accumulator 2 in Memory	129
STA1R	50XX	Store Accumulator 1 in Register	130
STA2R	51XX	Store Accumulator 2 in Register	130
XCH1	72XX	Exchange Accumulator 1 with Memory	131
XCH2	73XX	Exchange Accumulator 2 with Memory	131
XCH1R	70XX	Exchange Accumulator 1 with Register	132
XCH2R	71XX	Exchange Accumulator 2 with Register	132
ADD1	46XX	Add Memory to Accumulator 1	133
ADD2	47XX	Add Memory to Accumulator 2	133
ADD1M	42XX	Add Accumulator 1 to Memory	133
ADD2M	43XX	Add Accumulator 2 to Memory	133
ADD1R	44XX	Add Register to Accumulator 1	134
ADD2R	45XX	Add Register to Accumulator 2	134
SUB1	36XX	Subtract Memory from Accumulator 1	135
SUB2	37XX	Subtract Memory from Accumulator 2	135
SUB1M	32XX	Store Accumulator 1 Minus Memory in Memory	135
SUB2M	33XX	Store Accumulator 2 Minus Memory in Memory	135
SUB1R	34XX	Subtract Register from Accumulator 1	136
SUB2R	35XX	Subtract Register from Accumulator 2	136
XOR1	06XX	EXCLUSIVE OR Accumulator 1 with Memory	137
XOR2	07XX	EXCLUSIVE OR Accumulator 2 with Memory	137
XOR1M	02XX	EXCLUSIVE OR Accumulator 1 into Memory	137
XOR2M	03XX	EXCLUSIVE OR Accumulator 2 into Memory	137

MULTIPAC INSTRUCTION LIST -- continued

Mnemonic	Operation Code	Operation	Page
XOR1R	04XX	EXCLUSIVE OR Register with Accumulator 1	138
XOR2R	05XX	EXCLUSIVE OR Register with Accumulator 2	138
AND1	66XX	AND Accumulator 1 with Memory	139
AND2	67XX	AND Accumulator 2 with Memory	139
AND1M	62XX	AND Accumulator 1 with Memory into Memory	139
AND2M	63XX	AND Accumulator 2 with Memory into Memory	139
AND1R	64XX	AND Accumulator 1 with Register	141
AND2R	65XX	AND Accumulator 2 with Register	141
IOR1	26XX	INCLUSIVE OR Accumulator 1 with Memory	142
IOR2	27XX	INCLUSIVE OR Accumulator 2 with Memory	142
IOR1M	22XX	INCLUSIVE OR Accumulator 1 into Memory	142
IOR2M	23XX	INCLUSIVE OR Accumulator 1 into Memory	142
IOR1R	24XX	INCLUSIVE OR Accumulator 1 with Register	144
IOR2R	25XX	INCLUSIVE OR Accumulator 2 with Register	144
EQV1R	14XX	EQUIVALENCE Accumulator 1 with Register	145
EQV2R	15XX	EQUIVALENCE Accumulator 2 with Register	145
NEGR	30XX	Negate Register	146
MSTP	55XX	Multiply Step	146
LDLR	57XX	Load Register with Literal	148
ADLR	17XX	Add Literal to Register	148
MSKM	13XX	Replace Memory Through Mask	149
MSKR	11XX	Replace Register through Mask	149
NOP	00XX	No operation	150
INP	01XX	Input Through Mask	151
OUT	21XX	Output Accumulator 1	152
EINT	6076	Enable Interrupt	152
DINT	6077	Disable Interrupt	152

MULTIPAC INSTRUCTION LIST -- continued

Mnemonic	Operation Code	Operation	Page
SDMP	20XX (XX<40)	Select Data Memory Page	153
SPMP	20XX (XX≥40)	Select Program Memory Page	153
STPC	10XX	Store Program Counter in Register	153
MDI	12XX	Modify Next Instruction	154
JMP	16XX	Jump to Indexed Location	155
JMPR	56XX	Jump to Register Contents	155
		SKIP on Accumulator Condition (Various skips)	156
SKDR	31XX	SKIP on Decrementing Register	157
		Various shifts and cycles	158

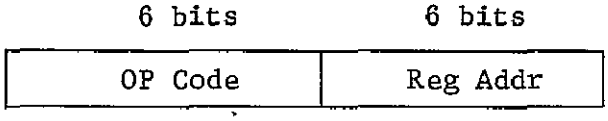
7.0 INSTRUCTION MANUAL

The logic modules are essentially small processing units which execute instructions from memory. Each logic unit has two 4-bit registers; one selects the memory to be used for program memory and the other selects the data memory. Each logic unit also has two scratch storage registers (called Accumulator 1 and Accumulator 2).

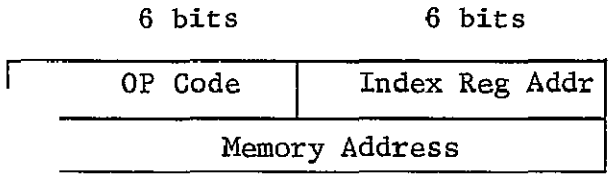
7.1 Instruction Formats

The instructions may use one or two words of memory, depending on whether or not memory is referenced. In general, they are of the following form:

Single Word Instruction:



Double Word Instruction:



Data words are 12 bits in length and are in two's complement notation.

Usually, the least significant bit of the OP code portion of the instruction defines whether Accumulator 1 or 2 is to be referenced.

The Register Address Field ("R" Field) generally specifies the register to be used for data in one-word instructions or the index register to be used for modifying the address field of two-word instructions. Since register 0 contains zero, "R" Field = 0 will produce no indexing for the latter case.

The logic unit addresses 64 register locations by the contents of the instruction "R" Field, or six lowest order bits. The first seven such locations are specifically assigned as follows:

Address	Register
0	Dummy register: Contents = 0
1	Accumulator 1 of Logic Unit
2	Accumulator 2 of Logic Unit
3	Input: Command Unit 1
3	Output: Telemetry Unit 1
4	Input: Command Unit 2
4	Output: Telemetry Unit 2
5	D/A Register 1
6	D/A Register 2

The remaining addresses in the first addressing section are nine. Address switching may optionally be included to expand the number of addresses in blocks of 16 to the maximum of 64. The unallocated register locations, up to 57, will normally be assigned to I/O registers, which makes the permissible I/O interface as large as 57 x 12 = 684 channels each way. These registers also serve the functions of index registers and provide scratch storage for the processor.

Accumulators 1 and 2 refer to the logic unit decoding the instruction. There is no way for one logic unit to address an accumulator in another logic unit. Since the accumulators are part of the Register addressing, they may be addressed with any of the Register Field instructions.

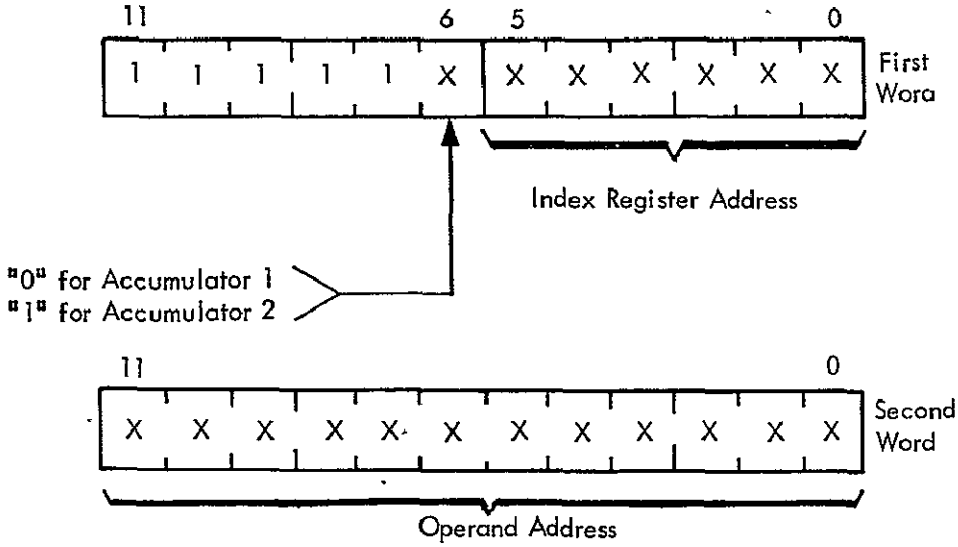
7.2 Arithmetic and Logical Instructions

All instructions which access memory are two-word instructions and require two memory cycles for their execution, assuming a data memory unit separate from that in which the program is stored. If only one memory unit is in use, i.e., if the contents of the program paging register and the data paging register are the same, the instruction cycle is automatically altered. Instructions which access program memory storage require three cycles for execution, halting the program counter for the necessary data access, and instructions which store in program memory require four cycles. All memory accesses are, in practice, indexed. Non-indexed instructions reference index register zero, and the contents of the dummy register RØ are hard-wired to present the number ZERO.

7.2.1 Instruction set:-- The available arithmetic and logical instructions are described in detail in the following pages.

LDA1	Load Accumulator 1 from Memory	76XX
LDA2	Load Accumulator 2 from Memory	77XX

Format:



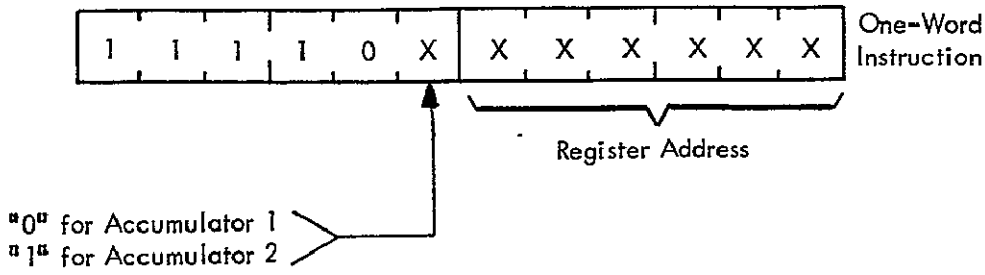
These instructions load one accumulator with the contents of the location specified by the sum of the operand address field and the contents of the index register. LDA1 loads Accumulator 1 and LDA2 loads Accumulator 2.

$$(\alpha + (R)) \Longrightarrow ACC_A \qquad [A = 1 \text{ or } 2]$$

LDA1R Load Accumulator 1 from Register 74XX

LDA2R Load Accumulator 2 from Register 75XX

Format:



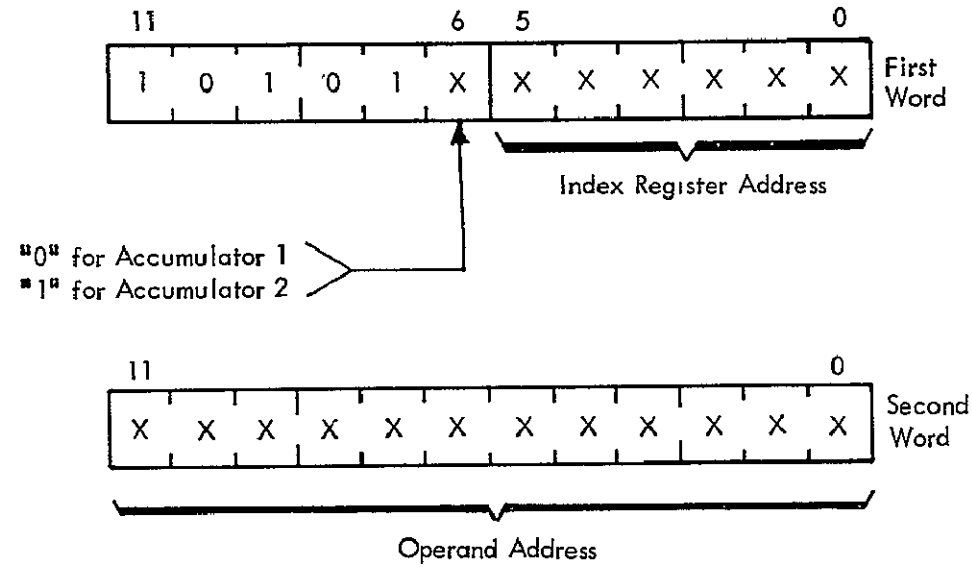
LDA1R, LDA2R load accumulator with the contents of the register specified by the Register Field. The register is unchanged. LDA1R loads Accumulator 1 and LDA2R loads Accumulator 2.

$(R) \Longrightarrow ACC_A$ [A = 1 or 2]

STA1 Store Accumulator 1 in Memory 52XX

STA2 Store Accumulator 2 in Memory 53XX

Format:



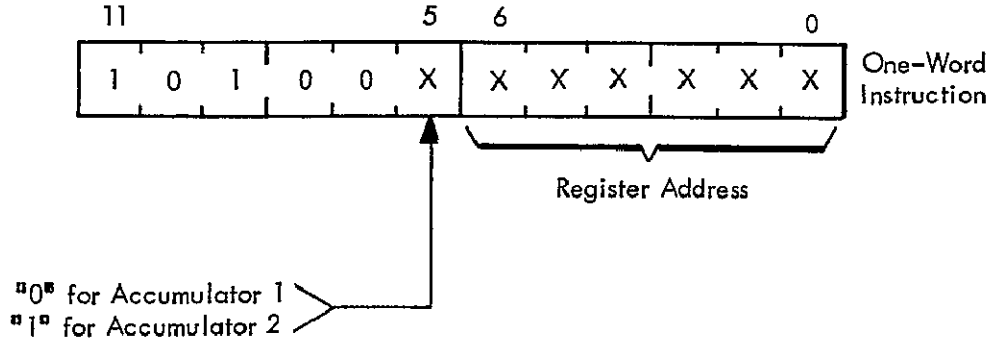
These instructions store the contents of one accumulator in the location of memory specified by the sum of the operand address field and the contents of the index register. The accumulators remain unchanged. STA1 stores Accumulator 1 and STA2 stores Accumulator 2.

$$\left(ACC_A\right) \Longrightarrow \alpha + (R) \qquad [A = 1 \text{ or } 2]$$

STA1R Store Accumulator 1 in Register 50XX

STA2R Store Accumulator 2 in Register 51XX

Format:



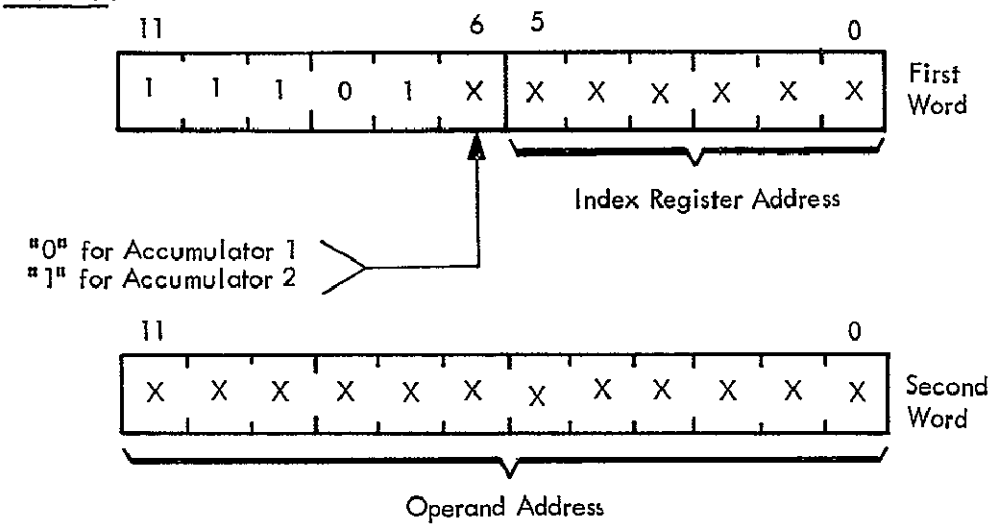
These instructions store the contents of one accumulator in the register specified. The accumulator remains unchanged unless specified in the Register Field. STA1R stores Accumulator 1 and STA2R stores Accumulator 2.

$$\left(ACC_A\right) \Longrightarrow R \qquad [A = 1 \text{ or } 2]$$

XCH1 Exchange Accumulator 1 with Memory 72XX

XCH2 Exchange Accumulator 2 with Memory 73XX

Format:



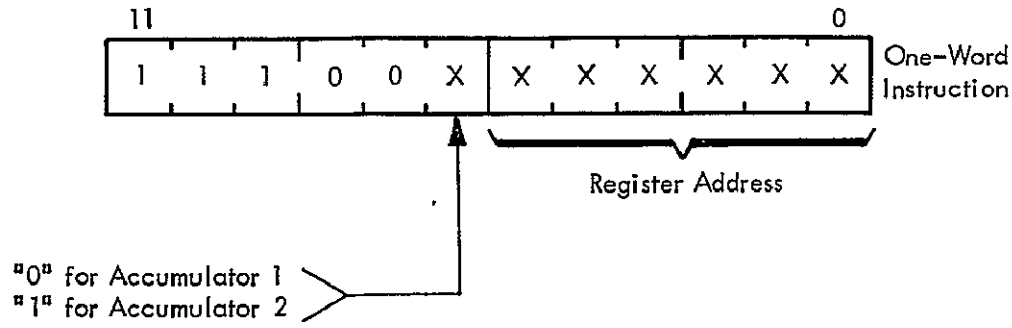
The contents of one accumulator and the location of memory which is the sum of the operand address field and the contents of the index register are interchanged. That location of memory will have the previous contents of the accumulator and the accumulator will have the previous contents of memory. XCH1 specifies Accumulator 1 and XCH2 specifies Accumulator 2.

$$\left.\begin{array}{l} (\alpha + R) \Longrightarrow ACC_A \\ \left(ACC_A\right) \Longrightarrow \alpha + (R) \end{array}\right\} \begin{array}{l} \text{simultaneously} \\ [A = 1 \text{ or } 2] \end{array}$$

XCH1R Exchange Accumulator 1 with Register 70XX

XCH2R Exchange Accumulator 2 with Register 71XX

Format:

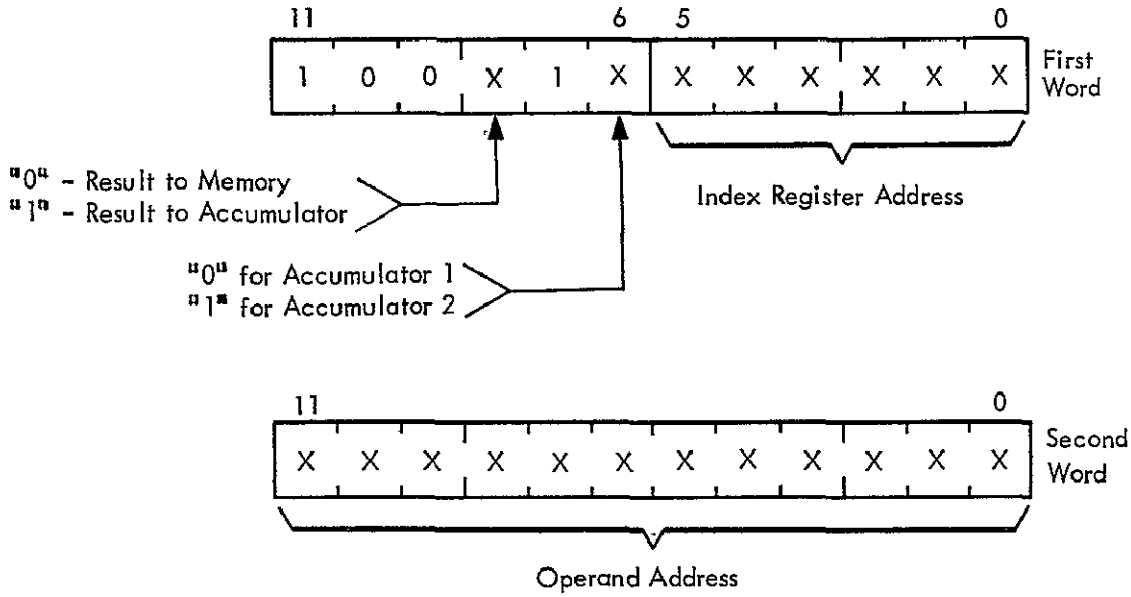


The contents of one accumulator and the contents of a register are interchanged. The register contains the old value of the accumulator and the accumulator the old value of the register. XCH1R specifies Accumulator 1 and XCH2R specifies Accumulator 2.

$$\left.\begin{array}{l}(R) \Longrightarrow ACC_A \\ (ACC_A) \Longrightarrow R\end{array}\right\} \text{simultaneously } [A = 1 \text{ or } 2]$$

ADD1	Add Memory to Accumulator 1	46XX
ADD2	Add Memory to Accumulator 2	47XX
ADD1M	Add Accumulator 1 to Memory	42XX
ADD2M	Add Accumulator 2 to Memory	43XX

Format:



The contents of the memory location addressed is added to the contents of one accumulator. For ADD1 and ADD2, the result replaces the accumulator contents, and for ADD1M and ADD2M, the result replaces the contents of memory. ADD1 and ADD1M reference Accumulator 1 and ADD2 and ADD2M reference Accumulator 2.

The address of memory is the sum of the contents of the index register and the operand address (second word).

ADD1 or ADD2:

$$\left(ACC_A\right) + (\alpha + (R)) \Longrightarrow ACC_A \qquad [A = 1 \text{ or } 2]$$

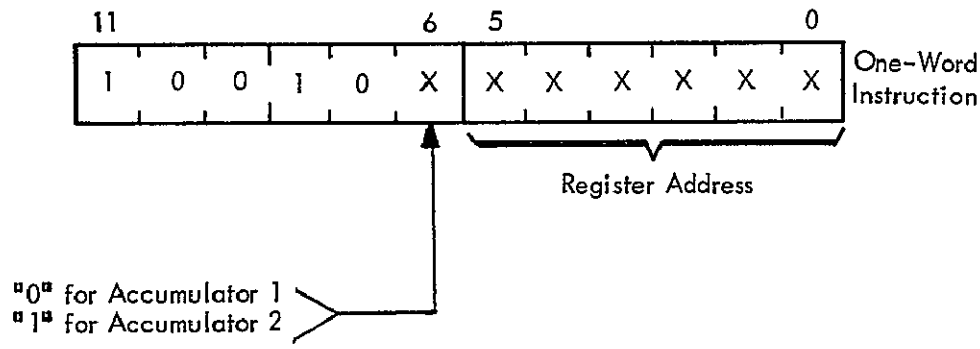
ADD1M or ADD2M:

$$\left(ACC_A\right) + (\alpha + (R)) \Longrightarrow \alpha + (R) \qquad [A = 1 \text{ or } 2]$$

ADD1R Add Register to Accumulator 1 44XX

ADD2R Add Register to Accumulator 2 45XX

Format:

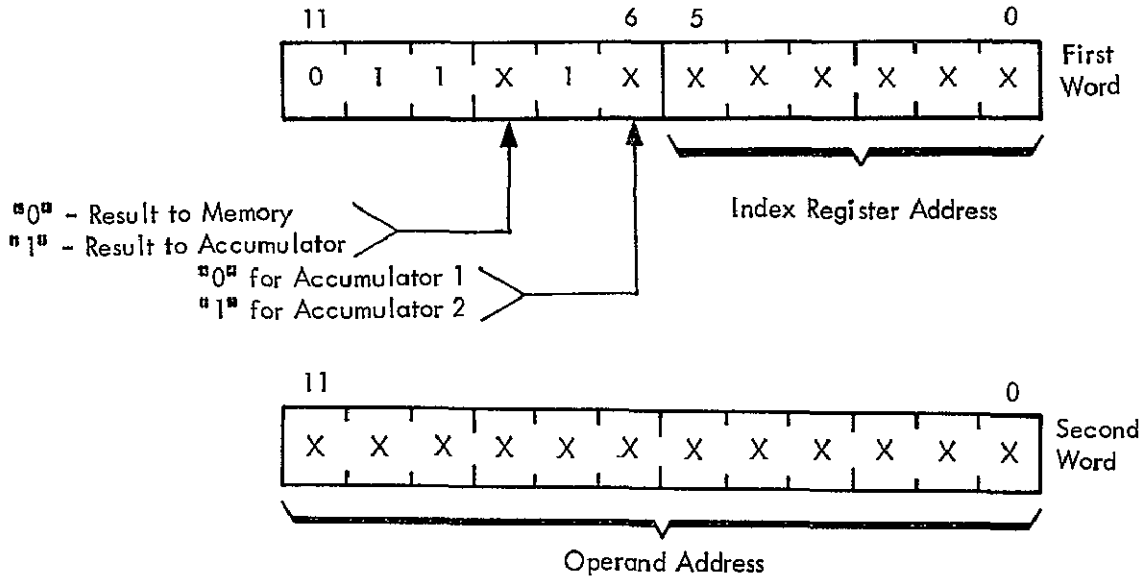


Add the contents of the register to one accumulator, leaving the results in the accumulator. ADD1R adds to Accumulator 1 and ADD2R adds to Accumulator 2.

$$\left(ACC_A\right) + (R) \Longrightarrow ACC_A \qquad [A = 1 \text{ or } 2]$$

SUB1	Subtract Memory from Accumulator 1	36XX
SUB2	Subtract Memory from Accumulator 2	37XX
SUB1M	Store Accumulator 1 Minus Memory in Memory	32XX
SUB2M	Store Accumulator 2 Minus Memory in Memory	33XX

Format:



Subtract the contents of memory addressed from Accumulator 1 if SUB1 or SUB1M or Accumulator 2 if SUB2 or SUB2M. For SUB1 or SUB2, replace the contents of the accumulator with the result. For SUB1M or SUB2M, replace the contents of memory with the result.

The location of memory addressed is the sum of the operand address and the contents of the index register.

SUB1 or SUB2

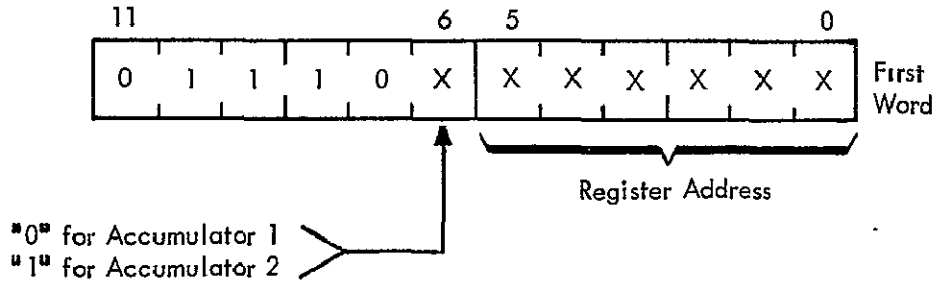
$$\left(ACC_A\right) - (\alpha + (R)) \Longrightarrow ACC_A \qquad [A = 1 \text{ or } 2]$$

SUB1M or SUB2M

$$\left(ACC_A\right) - (\alpha + (R)) \Longrightarrow \alpha + (R) \qquad [A = 1 \text{ or } 2]$$

SUB1R	Subtract Register from Accumulator 1	34XX
SUB2R	Subtract Register from Accumulator 2	35XX

Format:

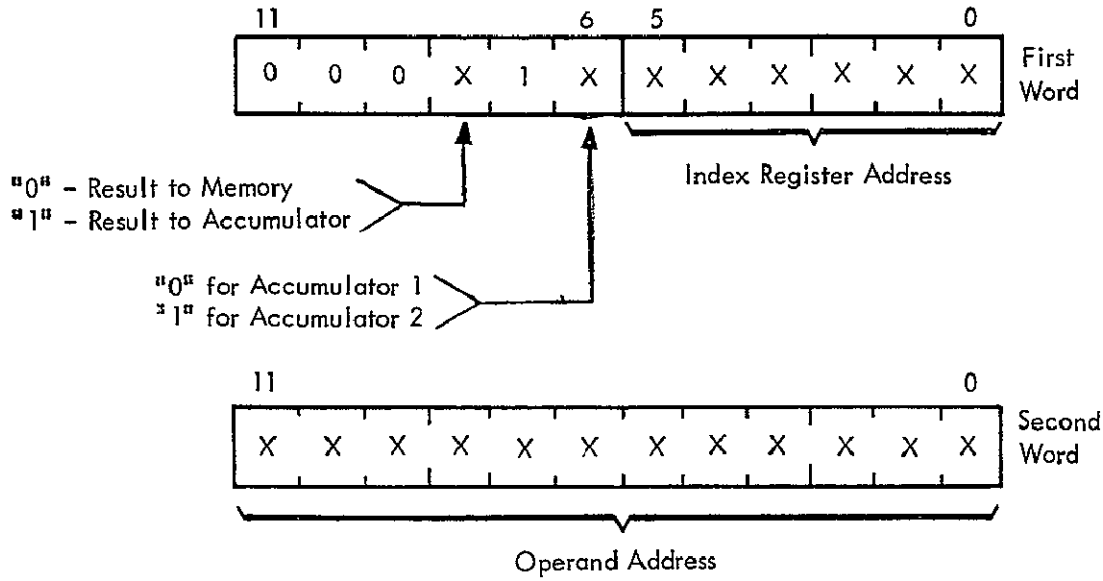


The contents of the register are subtracted from the contents of one accumulator, with the result replacing that accumulator contents. SUB1R references Accumulator 1 and SUB2R references Accumulator 2.

$$\left(ACC_A\right) - (R) \Longrightarrow ACC_A \qquad [A = 1 \text{ or } 2]$$

XOR1	EXCLUSIVE OR Accumulator 1 with Memory	06XX
XOR2	EXCLUSIVE OR Accumulator 2 with Memory	07XX
XOR1M	EXCLUSIVE OR Accumulator 1 into Memory	02XX
XOR2M	EXCLUSIVE OR Accumulator 2 into Memory	03XX

Format:



The EXCLUSIVE OR of the contents of the memory addressed and the contents of one accumulator is formed. For XOR1 and XOR2, the result replaces the contents of the accumulator, and for XOR1M and XOR2M, the result replaces the contents of the memory. XOR1 and XOR1M reference Accumulator 1. XOR2 and XOR2M reference Accumulator 2.

The memory location addressed is the sum of the operand address and the contents of the index register.

The EXCLUSIVE OR operation ($\oplus$) is performed on each bit position of the data independent of other bit positions by the following truth table.

Bit N of Accumulator	Bit N of Memory	Bit N of Result
0	0	0
0	1	1
1	0	1
1	1	0

XOR1 or XOR2

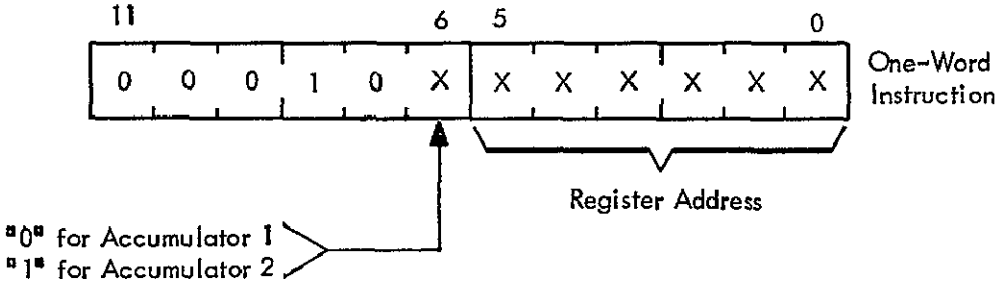
$$\left(ACC_A\right) \oplus (\alpha + (R)) \Longrightarrow ACC_A \qquad [A = 1 \text{ or } 2]$$

XOR1M or XOR2M

$$\left(ACC_A\right) \oplus (\alpha + (R)) \Longrightarrow \alpha + (R) \qquad [A = 1 \text{ or } 2]$$

XOR1R	EXCLUSIVE OR Register with Accumulator 1	04XX
XOR2R	EXCLUSIVE OR Register with Accumulator 2	05XX

Format:



The EXCLUSIVE OR of the contents of the register and the contents of one accumulator replaces the contents of that accumulator. XOR1R references Accumulator 1 and XOR2R references Accumulator 2.

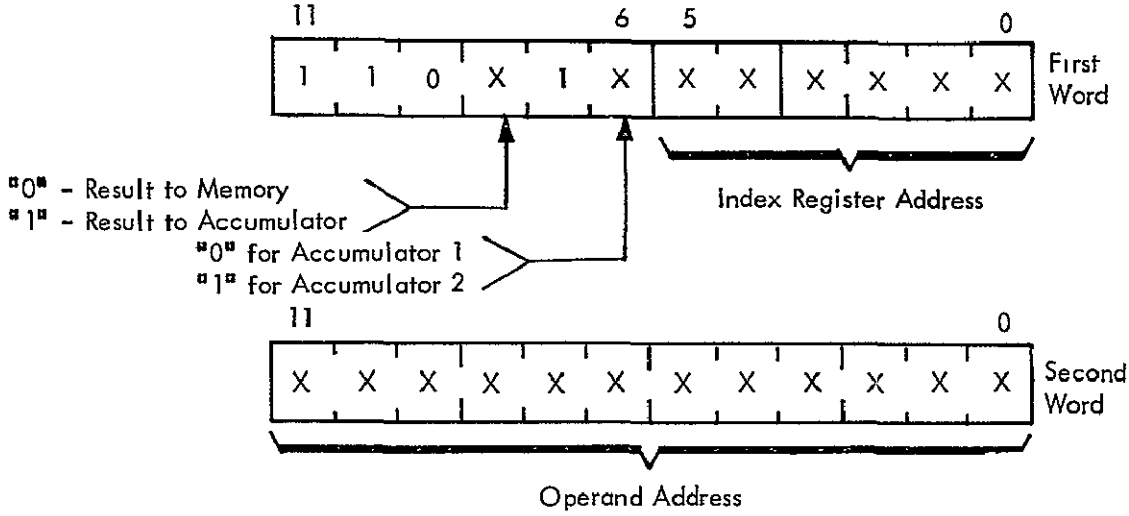
The EXCLUSIVE OR operation ($\oplus$) is performed on each bit position of the data independent of other bit positions by the following truth table.

Bit N of Accumulator	Bit N of Memory	New Bit N of Accumulator
0	0	0
0	1	1
1	0	1
1	1	0

$$\left(ACC_A\right) \quad \oplus \quad (R) \; => \; ACC_A$$

AND1	AND Accumulator 1 with Memory	66XX
AND2	AND Accumulator 2 with Memory	67XX
AND1M	AND Accumulator 1 with Memory into Memory	62XX
AND2M	AND Accumulator 2 with Memory into Memory	63XX

Format:



The logical AND of the contents of memory and the contents of one accumulator is formed. For AND1 and AND2, the results replace the contents of the accumulator, and for AND1M and AND2M, the results replace the contents of memory. For AND1 and AND1M, Accumulator 1 is referenced and for AND2 and AND2M, Accumulator 2 is referenced.

The loaction of memory addressed is the sum of the operand address (second word) and the contents of the index register.

The logical AND (∧) is performed on each bit position of the data independent of other bit positions by the following truth table:

Bit N of Accumulator	Bit N of Memory	Bit N of Result
0	0	0
0	1	0
1	0	0
1	1	1

AND1 or AND2

$$\left(ACC_A\right) \wedge (\alpha + (R)) \Longrightarrow ACC_A \qquad [A = 1 \text{ or } 2]$$

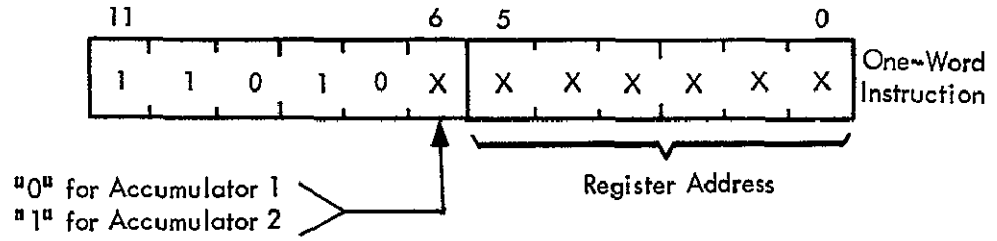
AND1M or AND2M

$$\left(ACC_A\right) \wedge (\alpha + (R)) \Longrightarrow \alpha + (R) \qquad [A = 1 \text{ or } 2]$$

AND1R AND Accumulator 1 with Register 64XX

AND2R AND Accumulator 2 with Register 65XX

Format:



The logical AND of the contents of the register and the contents of one accumulator replaces the contents of that accumulator. AND1R references Accumulator 1 and AND2R references Accumulator 2.

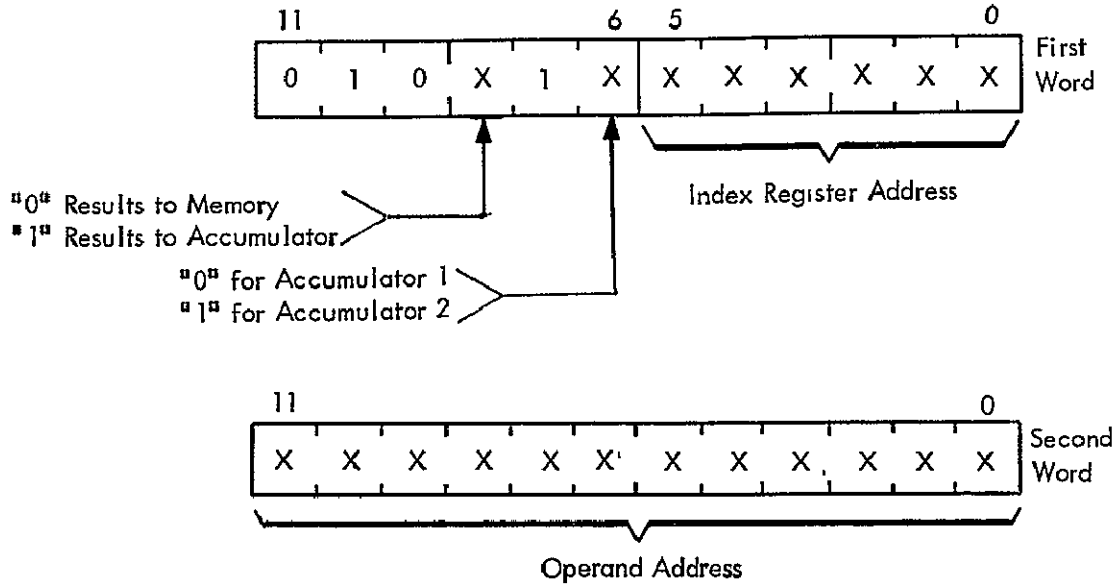
The logical AND ($\wedge$) is performed on each bit position of the data, independent of other bit positions according to the following truth table:

Bit N of Accumulator	Bit N of Register	Bit N of Accumulator
0	0	0
0	1	0
1	0	0
1	1	1

$$\left(ACC_A\right) \wedge (R) \Longrightarrow ACC_A \qquad [A = 1 \text{ or } 2]$$

IOR1	INCLUSIVE OR Accumulator 1 with Memory	26XX
IOR2	INCLUSIVE OR Accumulator 2 with Memory	27XX
IOR1M	INCLUSIVE OR Accumulator 1 into Memory	22XX
IOR2M	INCLUSIVE OR Accumulator 2 into Memory	23XX

Format:



The INCLUSIVE OR of the contents of memory with the contents of one accumulator is formed. For IOR1 and IOR2, the result replaces the contents of the accumulator, and for IOR1M and IOR2M, the result replaces the contents of memory. IOR1 and IOR1M reference Accumulator 1 and IOR2 and IOR2M reference Accumulator 2.

The location of memory addressed is the sum of the operand address (second word) plus the contents of the index register.

The INCLUSIVE OR (V) is performed bit by bit on each bit position of the data independent of the other bit positions according to the following truth table:

Bit N of Accumulator	Bit N of Memory	Bit N of Result
0	0	0
0	1	1
1	0	1
1	1	1

IOR1 or IOR2:

$$\left(ACC_A\right) \text{ V } (\alpha\ (R)\ ) \Longrightarrow ACC_A \qquad [A = 1 \text{ or } 2]$$

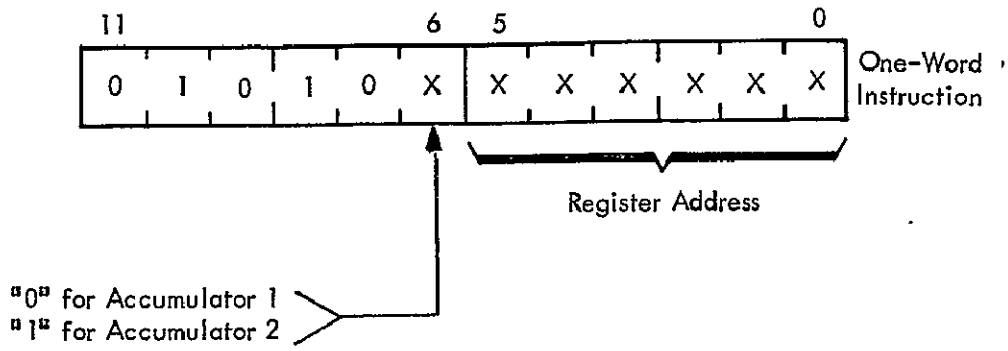
IOR1M or IOR2M:

$$\left(ACC_A\right) \text{ V } (\alpha + (R)\ ) \Longrightarrow \alpha + (R) \qquad [A = 1 \text{ or } 2]$$

IOR1R INCLUSIVE OR Accumulator 1 with Register 24XX

IOR2R INCLUSIVE OR Accumulator 2 with Register 25XX

Format:



The INCLUSIVE OR of the contents of the register with the contents of one accumulator replaces the contents of that accumulator. IOR1R references Accumulator 1 and IOR2R references Accumulator 2.

The INCLUSIVE OR (V) is performed bit by bit on each bit position of the data independent of the other bit positions according to the following truth table:

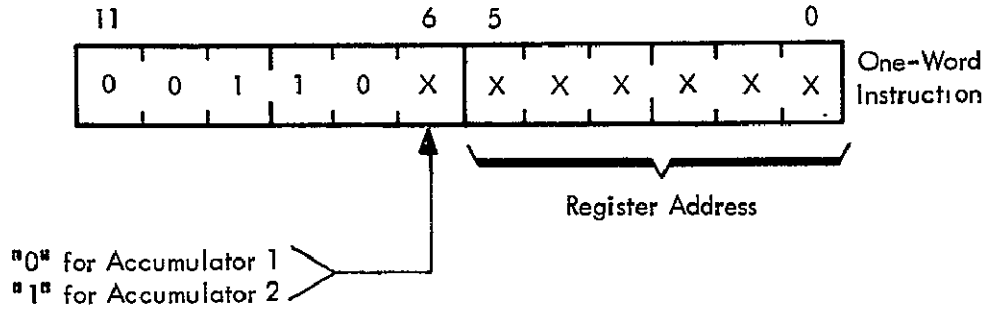
Bit N of Accumulator	Bit N of Register	Bit N of Accumulator
0	0	0
0	1	1
1	0	1
1	1	1

$$\left(ACC_A\right) \text{ V } (R) \Longrightarrow ACC_A \qquad [A = 1 \text{ or } 2]$$

EQV1R EQUIVALENCE Accumulator 1 with Register 14XX

EQV2R EQUIVALENCE Accumulator 2 with Register 15XX

Format:



The EQUIVALENCE of the contents of the register with the contents of one accumulator replaces the contents of that accumulator. EQV1R references Accumulator 1 and EQV2R references Accumulator 2.

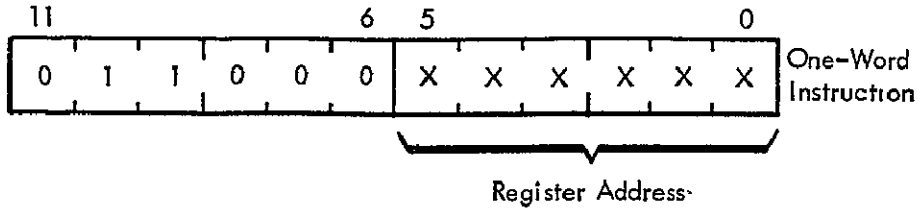
The EQUIVALENCE ($\odot$) is performed bit by bit on each bit position of the data, independent of other bit positions according to the following truth table:

Bit N of Accumulator	Bit N of Register	New Bit N of Accumulator
0	0	1
0	1	0
1	0	0
1	1	1

$(ACC_A) \odot (R) \Longrightarrow ACC_A$ [A = 1 or 2]

NEGR Negate Register 30XX

Format:

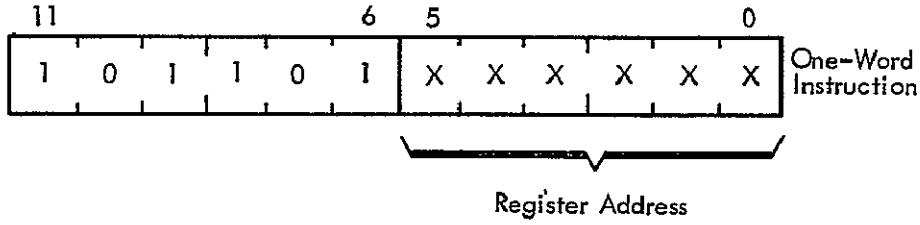


The contents of the register are replaced with the negation of the initial contents of the register. The negation is the two's complement of the initial contents.

(R) ⟹ R

MSTP Multiply Step 55XX

Format:



MSTP performs one step of the multiplication of the contents of the register by the contents of Accumulator 2, putting the result in double length register made up of Accumulator 1 and Accumulator 2.

Each MSTP consists of testing the low order bit of Accumulator 2 and if it is a "1", adds the contents of the register to Accumulator 1. Then, no matter what the lower order bit of Accumulator 2 is, Accumulator 1 and Accumulator 2 are shifted right one bit as a double length register, with the low order bit of Accumulator 1 shifting into the high order bit of Accumulator 2.

The multiply step (MSTP) instruction provides a convenient way of programming the multiply operation. Twelve such instructions will multiply the contents of a register (the multiplicand) by the contents of ACC2 (the multiplier) and leave the contents in the double-length register formed by ACC1 and ACC2. The multiplicand is treated as a signed number; however, the multiplier must be positive in this routine. If the multiplier might be negative, it could be tested and both multiplier and multiplicand negated before beginning the multiply. An alternate solution is to make a correction to the resultant.[a]

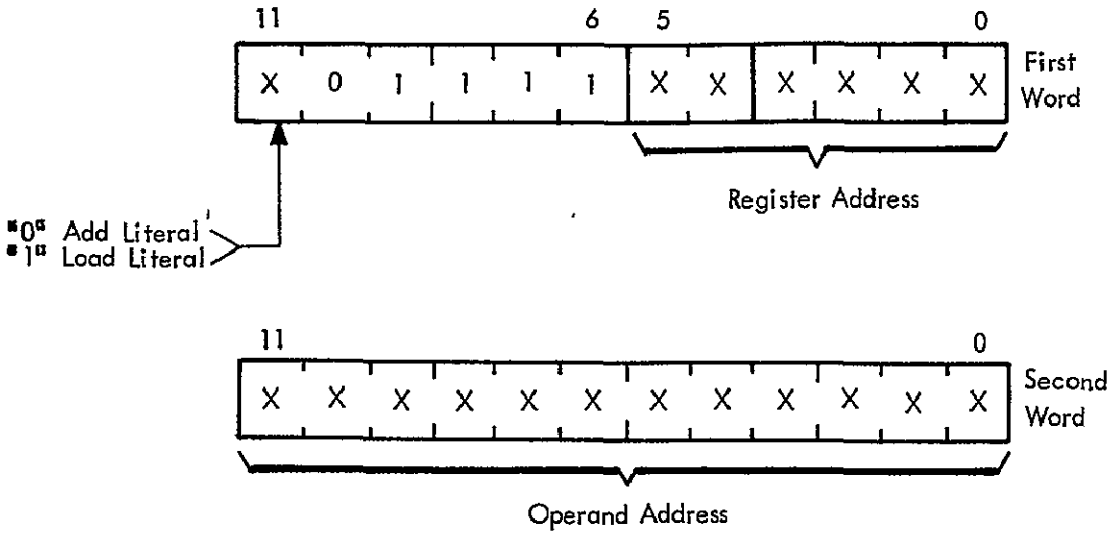
$$\text{ACC2} = 0: \left\{ \begin{array}{l} \text{ACC1}_n \Longrightarrow \text{ACC1}_{n-1} \\ \text{ACC1}_0 \Longrightarrow \text{ACC2}_{11} \\ \text{ACC2}_n \Longrightarrow \text{ACC2}_{n-1} \end{array} \right\} \quad 1 \leq n \leq 11$$

$$\text{ACC2}_0 = 1: \left\{ \begin{array}{l} (\text{R}) + (\text{ACC1}) \Longrightarrow \text{ACC1} \\ \text{ACC1}_n \Longrightarrow \text{ACC1}_{n-1} \\ \text{ACC1}_0 \Longrightarrow \text{ACC2}_{11} \\ \text{ACC2}_n \Longrightarrow \text{ACC2}_{n-1} \end{array} \right\} \quad 1 \leq n \leq 11$$

[a]The user should consult the literature. A particularly complete reference is "The Logic of Computer Arithmetic" by Ivan Flores (Prentice-Hall, 1963).

LDLR	Load Register with Literal	57XX
ADLR	Add Literal to Register	17XX

Format:



Load (LDLR) or add (ADLR) to the contents of the register the operand (second word).

LDLR:

$$\alpha \Longrightarrow R$$

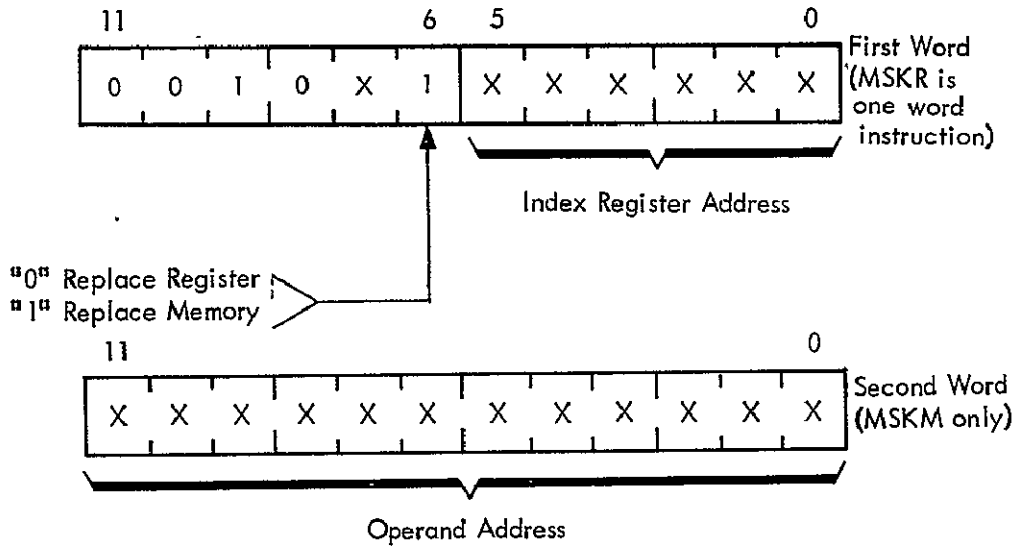
ADLR:

$$\alpha + (R) \Longrightarrow R$$

MSKM Replace Memory Through Mask 13XX

MSKR Replace Register Through Mask 11XX

Format:



Replace contents of memory or register with low order bits of Accumulator 1 through mask in Accumulator 2. Accumulator 2 is scanned from right to left (least significant bit to most significant bit). When the first "1" is encountered, the least significant bit of Accumulator 1 is transferred to memory or register in the same bit position as the "1" in the mask and Accumulator 1 is shifted right one bit. The scanning continues, and for every "1" in the mask, the above process is repeated.

If there are N "1"'s in Accumulator 2, then the net effect of these instructions is to replace the corresponding N bits of memory or register with the N least significant bits of Accumulator 1.

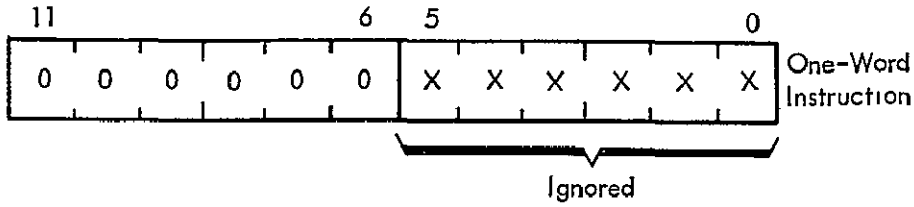
For MSKM, the location of memory is the sum of the operand address and the contents of the register. MSKR is a one-word instruction, and the contents of the register are replaced through the mask.

Example:

Accumulator 1:	0 0 0 0 0 1 1 1 0 1 0 1
Accumulator 2:	0 0 0 1 1 0 0 0 1 1 1 0
Register of Memory:	1 0 0 0 0 0 1 0 1 1 0 1
Final Register or Memory:	1 0 0 1 0 0 1 0 1 0 1 1
Final Accumulator 1:	0 0 0 0 0 0 0 0 0 0 1 1

NOP No Operation 00XX

Format:



No operation occurs with this instruction. The low order 6 bits are ignored.

7.3 Input/Output Instructions

The I/O Register consists of a shift register which receives and transmits data serially to the logic unit(s), and output buffer register into which the shift register transfers its contents upon receipt of an output code, and a set of parallel inputs to the shift register through which input I/O data is loaded upon receipt of an input code. These control codes are generated by the logic unit and are received and decoded by the Register Control Section. The control section then gates the register transfers as dictated by the codes. It also generates clock pulses to the receiving or outputting I/O devices to acknowledge the transfer of I/O data to or from them. In the case of serial transfers, for example, these serve as shift pulses shifting data into or out of the I/O device.

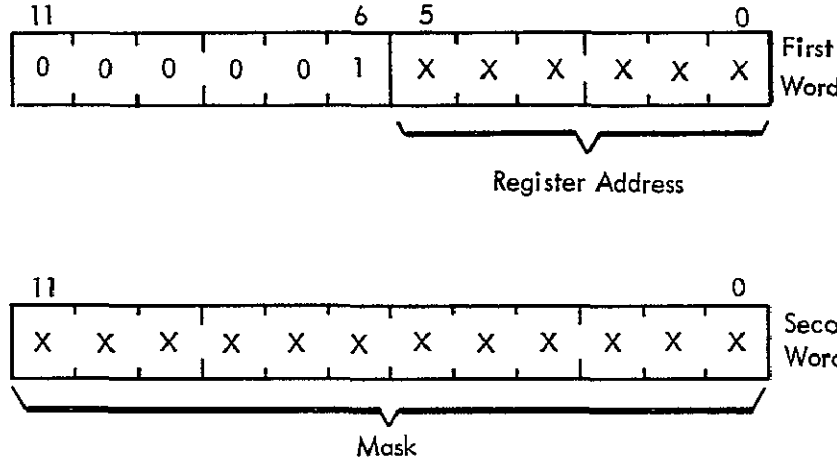
The INP instruction permits inputting any random bits determined by a mask into the low order end of ACC1 where they are right-justified. The OUT instruction permits loading data from ACC1 into a register and then directly into the output buffer register. These instructions also generate an acknowledge pulse to those I/O devices tied to the register.

Output information is not available to the processor from the interface itself and so a memory image of this information must be maintained. New information should, in general, be used to modify the memory image, and then the new image should be output to the buffers. An exception is data which is not to be retained at the interface, such as reset pulses or serial data.

7.3.1 Instruction set:-- The available input/output instructions are described in detail in the following pages.

INP Input Through Mask 01XX

Format:



The bit or bits specified by mask (second word) of the input interface of register R are inputted to the logic unit, shifting them left into the low order end of Accumulator 1.

There are twelve input interface bilevel lines tied to each bit position of each register. The INP instruction inputs all twelve lines at that register into the 12 bits of the register. Then, the new contents of the register is masked with the second word (mask) of the instruction.

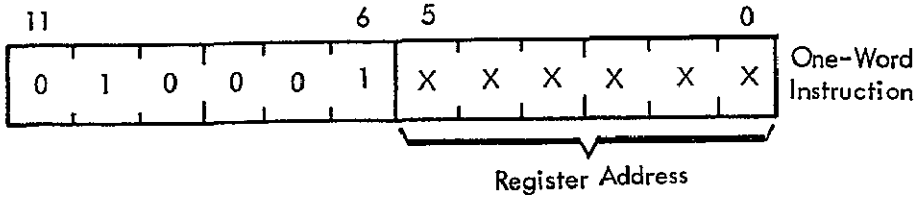
The new contents of the register and the mask are scanned from right to left (low order end to high order end). For each "1" in the mask, the corresponding bit of the register is shifted left into the low order end of Accumulator 1. For each "0", no transferring or shifting takes place.

Example:

Mask (second word):	0 0 0 1 **0** 0 0 1 **0** 0 1 **0**
Register after input strobe:	1 0 0 1 **0** 1 0 0 **0** 0 1 **0**
Accumulator 1:	0 0 0 0 0 0 0 0 0 0 0 0
New Accumulator 1:	0 0 0 0 0 0 0 0 0 1 0 1

OUT Output Accumulator 1 21XX

Format:

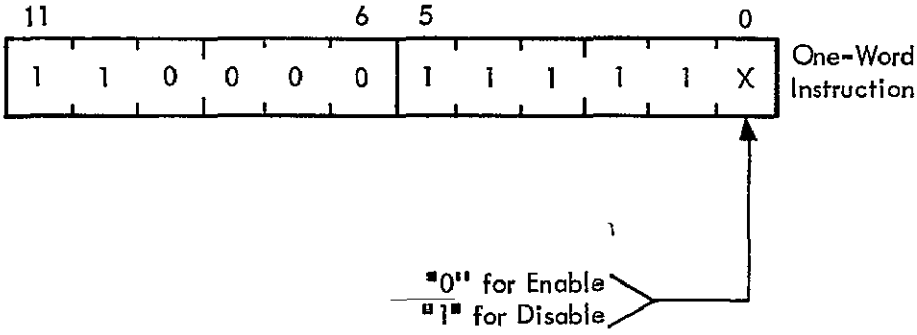


The contents of the register is replaced by the contents of Accumulator 1 and then the new contents of the register is transferred to the output buffer of the register. The 12-bit output buffer of each register is connected to the output interface.

EINT Enable Interrupt 6076

DINT Disable Interrupt 6077

Format:



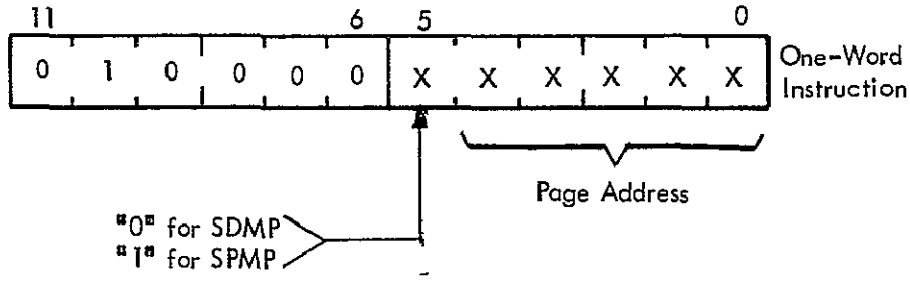
The interrupt flip-flop is enabled (for EINT) or disabled (for DINT).

7.4 Miscellaneous Instructions

SDMP Select Data Memory Page 20XX (XX < 40)

SPMP Select Program Memory Page 20XX (XX ≥ 40)

Format:

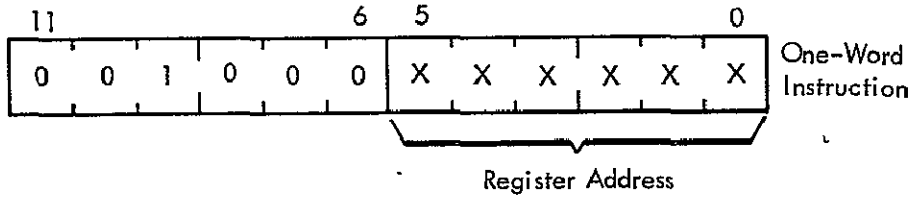


The page address is transferred to the program (for SPMP) or data (for SDMP) memory page register. If the page register is less than 5 bits, only the required least significant bits are transferred and the remainder ignored. When the program memory page register is set to zero by SPMP instruction, the logic unit is disabled and cannot be restarted except via the command override. Also, when the page is zero, the program counter (PC) is inhibited from incrementing, but it may be set or changed with a JMP or JMPR instruction.

Page Address ⟹ Page Register

STPC Store Program Counter in Register 10XX

Format:



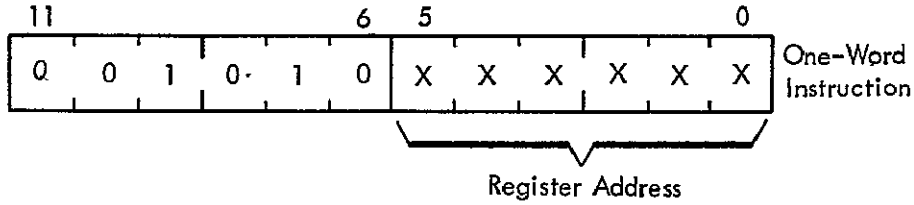
Replace contents of register with contents of program counter.

The store program counter (STPC) instruction provides a means of jumping to subroutines by storing the return address in a scratch register. Interrupt also uses this instruction stored in location zero. The interrupt hardware forces the program address to location zero and executes the instruction in that location before modifying the program counter. Thus, a STPC instruction in location zero will store the return address in a scratch register. The program counter is then forced to the number one, and the first instruction of the interrupt routine is taken from that location.

$$(PC) \Longrightarrow R$$

MDI Modify Next Instruction 12XX

Format:



Add the contents of the register to the next instruction before executing it.

The modify instruction (MDI) instruction provides a means of temporarily modifying the register address of an instruction during its execution. The instruction word in program memory is unchanged. This is useful in addressing specific I/O registers during general routines, operating indirectly through a register to a register. It is also useful as an execute instruction, executing the contents of the register as indexed by the program if the operation code is actually contained in the register, or in the case of skip conditions, if the condition code (register field) is in the register.

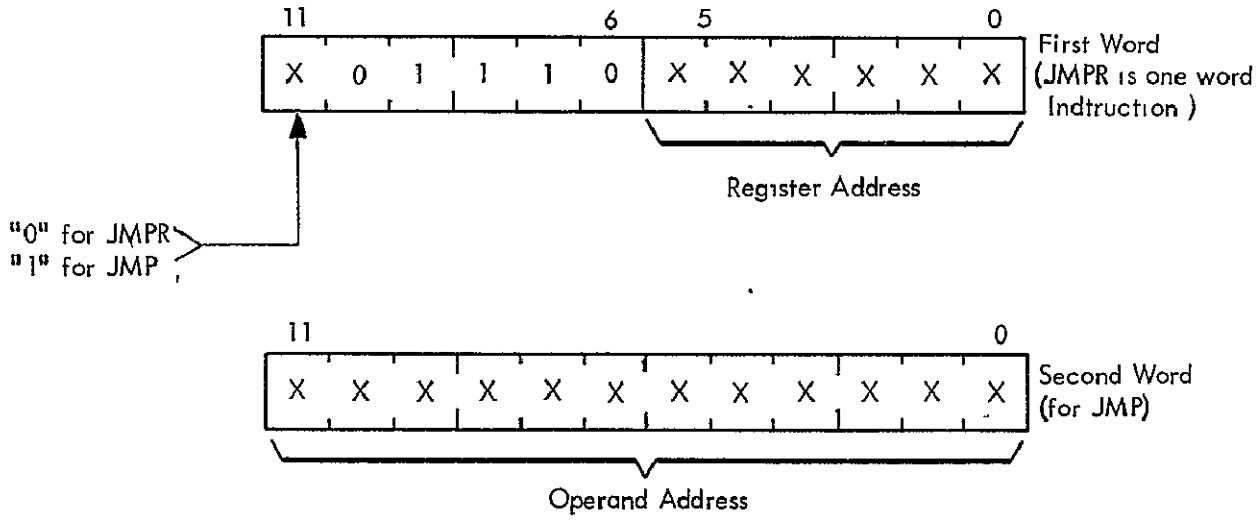
Since the register address field in shift is used for determining number of shift pulses, this MDI instruction can be used to shift by an amount contained in a register.

7.5 Branching Instructions

All jumps are unconditional and program branching is provided by the skip (SKP) instructions. These may be used to conditionally skip the two-word jump. One jump is also indexed inasmuch as the feature is readily available.

JMP	Jump to Indexed Location	16XX
JMPR	Jump to Register Contents	56XX

Format:

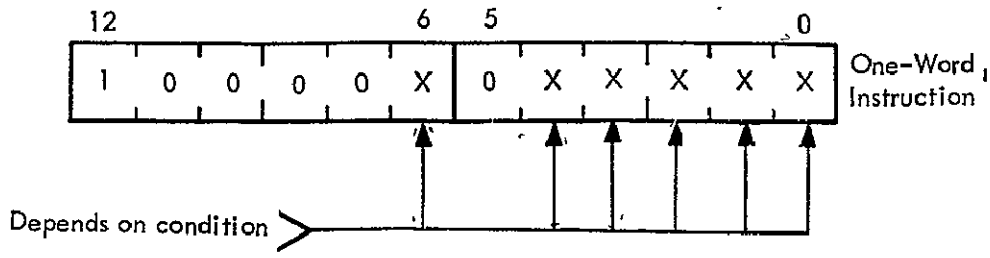


For JMP, the next instruction is taken from the sum of the operand address plus the contents of the register.

For JMPR, which is a one-word jump, the next instruction is taken from the location stored in the register. This instruction allows return from subroutines or interrupt if the program counter had been previously stored in the register with an STPC instruction.

SKIP on Accumulator Condition

Format:



Each of these instructions tests a condition of the two accumulators, and if the condition is met, skips the next two locations in program memory. If the condition is not met, the next instruction (location) is executed.

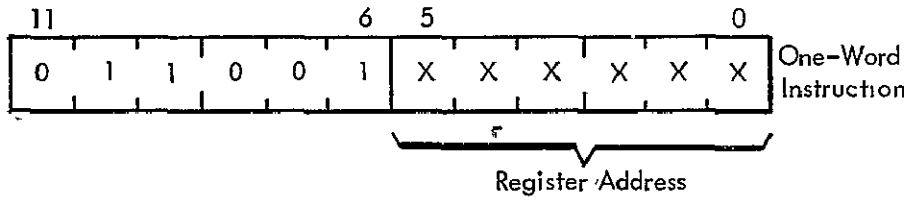
Skip if	OP Code
ACC1 = ACC2	4076 or 4077 or 4176 or 4177
ACC1 > ACC2	4054 or 4055
ACC1 < ACC2	4154 or 4155
ACC1 ≠ ACC2	4056 or 4057 or 4156 or 4157
ACC1 ≤ ACC2	4074 or 4075
ACC1 ≥ ACC2	4174 or 4175
ACC1 ≤ 0	4015
ACC2 ≤ 0	4115
ACC1 > 0	4035
ACC2 ≥ 0	4136
ACC1 ≥ 0	4114
ACC2 ≥ 0	4014
ACC1 < 0	4134
ACC2 < 0	4034
ACC1 = 0	4017 or 4116
ACC2 = 0	4117 or 4016
ACC1 ≠ 0	4037 or 4136
ACC2 ≠ 0	4137 or 4036
Overflow	4020 or 4120
No overflow	4000 or 4100

Bit N of ACC1 = 0	4000 + N	$1 \leq N \leq 14$
Bit N of ACC2 = 0	4100 + N	$1 \leq N \leq 14$
Bit N of ACC1 = 1	4020 + N	$1 \leq N \leq 14$
Bit N of ACC2 = 1	4120 + N	$1 \leq N \leq 14$

The last four test each bit of an accumulator separately. The bits are numbered from right to left (least significant to most significant) starting with 1. Thus, the most significant bit is 14 (octal) or 12 (decimal).

SKDR Skip on Decrementing Register 31XX

Format:



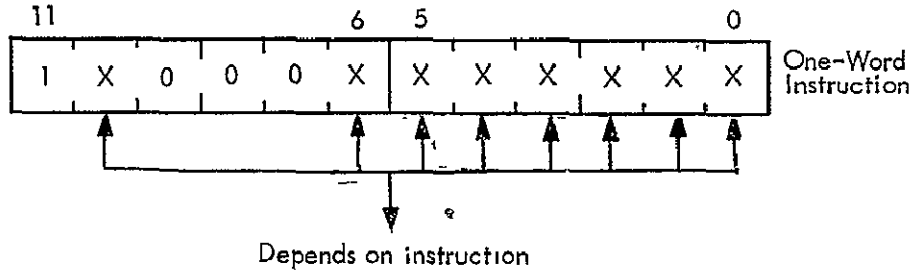
This instruction decrements the register, and if the result is zero, skips the next two memory locations. The decrementing of the register always takes place.

$$(R) - 1 = 0: \begin{cases} (R) - 1 \Longrightarrow R \\ (PC) + 2 \Longrightarrow PC \end{cases}$$

$$(R) - 1 \neq 0: \quad (R) - 1 \Longrightarrow R$$

7.6 Shifting Instructions

Format:



These instuctions shift one or both accumulators right or left, arithmetically, logically or cyclicly. They can shift by 0 through 13 (octal) or 11 (decimal) shifts.

Mnemonic	Description	OP Code
SHRL1	Shift Accumulator 1 right logically N bits	6060 + N
SHRL2	Shift Accumulator 2 right logically N bits	6160 + N
SHRA1	Shift Accumulator 1 right arithmetically N bits	6000 + N
SHRA2	Shift Accumulator 2 right arithmetically N bits	6100 + N
SHL1	Shift Accumulator 1 left N bits	6020 + N
SHL2	Shift Accumulator 2 left N bits	6120 + N
CYC1	Cycle Accumulator 1 right N bits	6040 + N
CYC2	Cycle Accumulator 2 right N bits	6140 + N
DLSRL	Shift Accumulators 1 and 2 together right logically N bits	4040 + N
DLSRA	Shift Accumulators 1 and 2 together right arithmetically N bits	4140 + N
DLSHL	Shift Accumulators 1 and 2 together left N bits	4160 + N
DLCYC	Cycle Accumulators 1 and 2 together right N bits	4060 + N

When shifting right logically (SHRL1, SHRL2, DLSRL), the new high order bits are filled with zeros. When shifting right arithmetically (SHRA1, SHRA2, SLDRA), the new high order bits are filled with the sign bit. When shifting left (SHL1, SHL2, DLSHL), the new low order bits are filled with zeros.

For double length shift rights (DLSRL, DLSRA) and cycle (DLCYC), the low order bit of Accumulator 1 is shifted into the high order bit of Accumulator 2. For double length shift left (DLSHL), the high order bit of Accumulator 2 is shifted into the long order bit of Accumulator 1.

For single length cycles (CYC1 and CYC2), the low order bit of the accumulator is shifted into the high order bit. For double length cycle (DLCYC), the low order bit of Accumulator 2 is shifted into the high order bit of Accumulator 1.

8.0 PROGRAMMING

8.1 Typical Subroutines

Several pieces of the required operating system have been coded in order to achieve three goals: (1) develop an approximation of the timing involved, (2) discover difficulties in programming imposed by the design, and (3) develop useful techniques for circumventing various design limitations.

These routines were first programmed using the order code of the previous version of MULTIPAC (see MULTIPAC Research Report).[1] Those programs helped to point out deficiencies in the previous design which led to the present LSI design. These routines have been reprogrammed (see Tables 13, 14, and 15) using the new order code resulting in about a third the number of words and execution time.

8.1.1 A/D conversion subroutine.-- Table 13 is an analog-to-digital conversion subroutine. The semicolon denotes comments and the colon denotes a program label. Those signals of the experiments which are to be converted are fed into an anlog comparator (difference circuit) at the experiment, the other input of which is the output of one of the digital-to-analog registers in MULTIPAC. This circuit sends a bilevel signal to the computer telling it whether or not the signal is higher or lower than the D/A reference signal. Using a binary successive approximation technique, the conversion routine determines the analog value. First, it tests for less than or greater than half full range. Then, it tries half the resulting range of the first test and continues to halve the range until it obtains the value.

8.1.2 Inputting subroutine.-- Table 14 is a subroutine for inputting serial data from an experiment. This simple routine first determines the address of the specific routine from a table and then, for this case, inputs a serial stream of n bits.

8.1.3 Formatting subroutines.-- Table 15 is a subroutine for formatting and then outputting one 12-bit telemetry word. This routine would be called when the telemetry signals that it needs a new word. In contrast to previous assumptions, parity (or error correcting codes) is added by the telemetry hardware. A flow chart of this routine is given in Figure 35.

8.1.4 Timing.-- The routines chosen represent the most important and most time consuming of the primary tasks of the CDS. Additional time must be allotted for the executive routine and the remaining tasks. The results for the primary tasks are summarized here.

TABLE 13

A/D CONVERSION ROUTINE

```
;RINDEX = Register used as index register
;N      = number of bits to be converted
;RDA    = one of the D/A registers
;RDEV   = I/O register to which experiment's comparator
          returned
;DSEL   = 12-bit mask for selecting current input channel
          of RDEV
; All numbers in octal notation

        LDLR    N,RINDEX        ;number of bits to index register
        LDLR    600, ACC2       ;mask used for outputting
        LDA1R   0               ;clear accumulator 1
        OUT     RDA             ;clear D/A, signal experiment
LOOP:   ADLR    1,ACC1          ;one to least significant bit
        MSKR    RDA             ;output two bits to D/A
        SHRL2   1               ;move mask for next trial
        INP     DSEL, RDEV      ;input return from comparator
        SHL1    1               ;move input bit left, 1 added at LOOP
        SKDR    RINDEX          ;loop back for total at N times
        JMP     LOOP
         .                      ;result in RDA
         .
```

Timing: Overhead = 6 cycles = 90 μs

Loop = 10n cycles, = 150n μs, where n = number of converted bits

Variable significant timing = 6 + 10n cycles

= 90 + 150n μs

n = 5 t = 840 μs
n = 8 t = 1290 μs

TABLE 14

INPUTTING ROUTINE

```
;DNUMB   =  device number
;DEVINP  =  table of inputting routine addresses
;RRTN    =  register which contains return address to
            routine which called READ
READ:       LDLR    DNUMB, RX   ;device number to index
            LDA2    DEVINP, RX  ;address of routine to ACC2
            JMPR    ACC2        ;jump to proper subroutine (DEVN)
             .
             .

;NBITS   =  number of bits to read in
;RINDEX  =  index register
;RDEV    =  I/O register with desired input
;DSEL    =  mask for selecting proper input channel
DEVN:       LDLR    NBITS, RINDEX   ;number of bits to index
            LDA1R     0             ;clear accumulator 1
DEVN1:      INP     DSEL;RDEV       ;input bit
            SKDR    RINDEX          ;loop back for all bits
            JMP     DEVN1
            JMPR    RRTN            ;return with result in ACC1
```

Timing:

Overhead = 11 cycles

Read loop = 5n cycles ;n = no. bits to read

Total time = 11 + 5n cycles

= 165 + 75n μs

n = 5 t = 540 μs
n = 8 t = 765 μs
n = 12 t = 1065 μs

Range: $540 \leq t \leq 1065$ μs

TABLE 15

OUTPUTTING ROUTINE
(Outputs one 12-bit telemetry word)

```
;TBUF  = address of data table
;OBUFI = current index into TBUF
;TBUF(OBUFI) is next data and TBUF(OBUFI + 1) is number of bits in data word
;ROBUFI = scratch register to hold OBUFI
;RBLEFT = scratch register to add number of bits left to output
;RWDCNT = scratch register to add number of bits in data word
;RTLM = one of the telemetry registers
;RRTN = scratch register holding return address
;all numbers in octal notation

TLMOUT  LDA1   OBUFI           ;data word index to ROBUFI
        STA1R  ROBUFI
        LDLR   RBLEFT,14       ;number of bits to output
        LDA1R    0             ;clear accumulator 1
TLOOP:  LDA2R  TBUF+1, ROBUFI  ;bit length of next data
        STA2R  RWDCNT          ;store in RWDCNT
        SUB2R  RBLEFT          ;word count minus bits left
        SKIP   (ACC2≤0)        ;test for too many bits...
        JMP    TOVER           ;over, go to TOVER
        STA2R  RBLEFT          ;OK, update bits left
        NEGR   RBLEFT
        LDA2   TBUF,ROBUF1     ;get data word
        MDI    RWDCNT          ;shift whole word...
        DLSRL       0          ;into accumulator 1
        ADLR   ROBUFI,2        ;update table index...
        LDA2R  ROBUFI          ;and store away
        STA2   OBUFI
        LDA2R  RBLEFT          ;check if any bits left..
        SKIP   (ACC2=0)
        JMP    TLOOP           ; yes, go back for more
        JMP    TEXIT           ;no, go to TEXIT
```

TABLE 15.-- Continued

OUTPUTTING ROUTINE

```
TOVER:  STA2   TBUF + 1, ROBUFI    ;amount over = number of bits
                                     next time
        LDA2   TBUF, ROBUFI        ;get data word
        MDI    RBLEFT              ;shift only those needed...
        DLSRL    0                 ;into accumulator 1
        STA2   TBUF, ROBUFI        ;store remainder back in table

TEXIT:  OUT    RTLM                ;output result to telemetry
        JMPR   RRTN                ;return to calling routine
```

TIME: Overhead (TLMOUT up to TLOOP and TEXIT to end) = 14 cycles

each word not overflowing (TLOOP up to TOVER) = 19n cycles, n = no. of times

last word (TLOOP thru TOVER up to TEXIT) = 13 cycles

Total time = 27 + 19n cycles

or $t = 405 + 285n$ μs

n = 1 (word length averaging 12 bits) $t = 690$ μs

n = 2 (word length averaging 8 bits) $t = 975$ μs

n = 3 (word length averaging 6 bits) $t = 1260$ μs

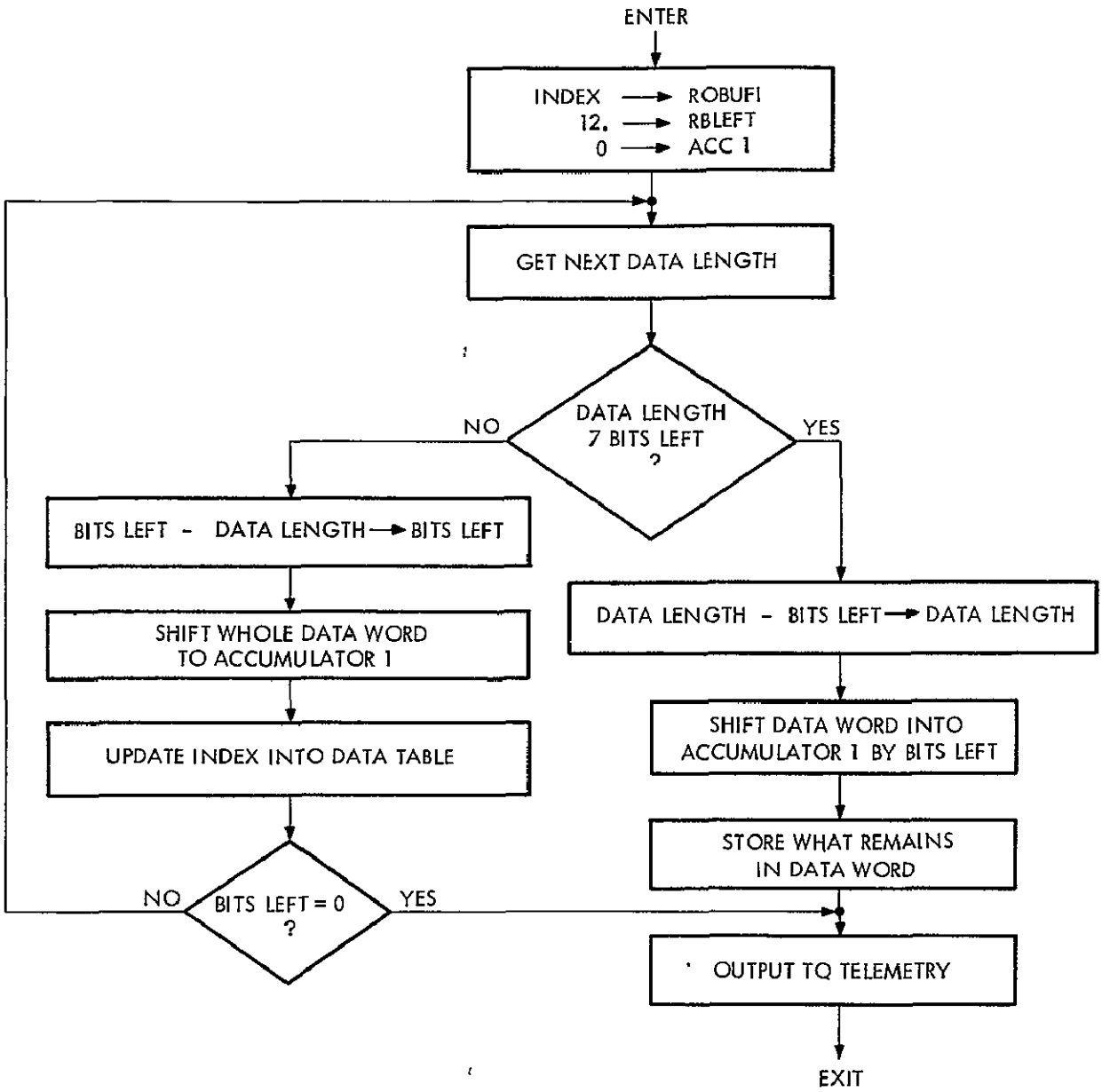


Figure 35. Outputting Routine Flow Chart

	A/D Conversion (μsec)	Inputting (μsec)	Formatting (μsec)
Time Evaluation	150n + 90	75n + 165	--
n = 5 bits	840	540	1260
n = 8 bits	1290	765	975
n = 12 bits		1065	690

If the above tasks of A/D conversion, input, and telemetry buffering are the only tasks desired, then the basic time, not including interconnecting overhead for each device, for an average 8 bits per device (analog or digital) would be 2.3 to 2.8 msec/dev (assuming 500 microseconds used in interconnection overhead). At 2.5 msec/dev, 400 devices per second is saturation. That results in 3200 bits/second to the telemetry. To gain more speed, the process would have to be divided over several logic units.

8.2 Communication Between Processes

A typical MULTIPAC central data system will be configured to operate three processes simultaneously, i.e., one with each logic unit. The most likely division of these three tasks will be data formatting and output to the telemetry unit, inputting and converting data from the experiments, and performing data reductions on the experimental data. Each process will take a logic unit, two memories and one to four registers, except for the inputting which must be able to sample all the registers which have input information on the register input channels. Since all three processes operate on the same data base, there must be some communication between the processes. A process may need to know when another process has data for it or needs new data, and when data is ready, some means of transfering the data from one process to another must be available. The easiest way to transfer large amounts of data between processes is to switch memories. For example, when the inputting routine has inputted enough data to be used by either the data reduction process or the outputting process, it would be desirable to have some means to notify the other process that the data is ready so that this process can switch its data memory page to the new data. For one or two words transfers, a transfer via a register is probably easiest since all logic units can address all registers.

All methods of data transfer require some method of flagging one process by another process. The best technique to perform this type of communication will be to use some program specified register as a flag register.

In MULTIPAC, when two logic units are accessing the same register, both logic units will be able to read the register data, but the data going back to the register is ORed. The registers, when shifted, do not recycle their data, but the logic unit which is reading the register rewrites the contents of the register. As long as both logic units are

both reading, they will be rewriting the same data. If one, or both, are storing new data in the register, then the logical OR of the outputs of the two logic units will result.

To circumvent this problem of simultaneous register use, the following programming rule must be followed. If a process is writing a "0" into a flag register, it then must write the "0" for two consecutive instruction times. Except for this case of writing "0", a flag register may not be addressed by two consecutive instructions. Since no two consecutive reads are permitted to occur, the write will eventually win out.

One register would be used for the flags of two processes. Each process will use one-half of the flag register for its flags. One process sets flags at its portion of the register as signals to the other and reads the other portions to determine what the other processes are doing.

The basic procedure is to set a busy flag for a module if it is not already in use by another logic unit and proceed to use it. To avoid the conflict of two processes finding a module free and proceeding in unison to use it, the following procedure will be followed: Let us suppose processor P1 wants to use a module. Then, if R is the flag register, P1 can proceed as follows:

(1) Read R and check if busy flag is on. If so, wait; if not, proceed.

(2) Pause, then write R with P1 busy flag on (1 = busy).

(3) Read R and see if P2 busy flag is now on (in case P2 read R before P1 wrote R). A pause before reading R is needed in case P2 is setting some other bit to zero). If P2 busy flag is still off, then proceed to use module.

(4) If P2 busy flag is on, then P1 has two choices:

    a. if P2 has priority, then wait until module is free again;

    b. If P1 has priority, then proceed because P2 will wait.

(5) When P1 is finished with module, it must turn its busy flag off. R must be written twice with the flag = 0 because of the P2 rewrite cycle.

The above procedure can be used between the process which is inputting data from experiments and the process which is formatting the data (assuming no data reduction) and outputting to the telemetry register. Two different data memories will be used to buffer the input data. The inputting routine will use one buffer and the outputting routine will use the other buffer. When they are both finished using their buffer, the data memories will be swapped and the process will continue.

The inputting routine will set a busy flag before it fills one buffer. When it is finished, it will turn off the busy flag, set a second flag to signify the data is ready, and then, using the procedure above, will check to see if the second buffer is free for use. The formatting routine will read the data-ready flag set by the inputting routine to tell when the data is ready and sets its busy flags to notify the other routine what it is doing. Since each of the two buffers is in a different memory, then the transfer is a simple switch of the data memory page register.

8.3 Data Reduction

Data reduction programming will have to be done by the experimeter. To the experimenter. To the experimenter various subroutines can be made available, such as histogram or fourier analysis. Data reduction techniques make some assumptions about the nature of the data. Consequently, it may be desirable to add data reduction techniques after the spaceprobe has gathered data. These programs can be checked out by the ground base computer and then transmitted to the spacecraft via the command link.

8.3.1 Histograms or quantiles.-- Histograms or quantiles take very little space for program storage, probably 100 to 300 words of program memory. Data storage, on the other hand, will most likely require 1000 words for each experimental line analyzed. Probability theory for normally distributed data and single quantiles state that the square of the mean deviation will be 1.57 divided by the number of samples, and for two quantiles, 0.767 divided by the number of samples. Thus, 1000 samples seems a reasonable amount to keep the error to a few percent. The determination of the optimal quantiles requires the knowledge of the density function of the underlying population whose parameters are being estimated. Thus, the results will not be optimal when applied to populations whose densities depart from that assumed in finding the quantiles.

8.3.2 Digital filters.-- Probably the best implementation of digital filters is to cascade recursive filter sections using different equations. Cascaded sections require the least accuracy of the data word and are the simplest to implement. The canonical form of difference equations is generally preferred in terms of ensuring against noise due to truncation and round off effects. Recursive filters require very little program storage and data storage. A very complicated cascaded filter can probably be implemented in a few hundred words of program storage and 10 to 20 data words.

8.3.3 Spectral analysis.-- The most generally useful program technique for spectral analysis is the fast fourier transform (FFT) which can be programmed in less than a thousand words of program space. The data space is a function of the size of the transform and is approximately

twice the number of points in the transform. Half the data storage is for the data points themselves, a quarter for the cosine table and the remainder for miscellaneous constants. For example, a 512-point transform will take about a thousand words of data storage. A 512-point transform can be considered as 512 simple single-pole filters spread evenly across the bandwidth over a time span of 512 consecutive samples.

Reliable spectral estimates are possible only if the experimenter has a rough idea of the actual spectrum being estimated. Such knowledge will then enable an intelligent choice of the number of samples and the bandwidth of the bandpass filter preceding the transform. The sampling frequency must be high enough to minimize possible aliasing errors, a selection that demands the knowledge of the spectrum shape. Quite often the spacecraft instruments have a well-defined bandwidth so that the selection of the necessary sampling rate (Nyquist rate) is straightforward.

8.3.4 Usage of data reduction techniques.-- During the first year of this study, some analysis was made of what experiments might use these data reduction algorithms. The advantage of a computer in achieving data compression has been a primary concern.

8.3.4.1 Neutron detector.-- The interface between the neutron detector and the CDS is shown in Figure 36. The bit requirement for the measurements alone, exclusive of timing information, amounts to 208 bits per second for the neutron detector. One way of reducing this requirement is to accumulate data over only a few hours during a day or to sample less frequently, thereby lowering the average bit rate. This may be undesirable since it sacrifices what might be valuable information. Use of logarithmic counters would immediately reduce the bit rate.

Processing alternatives:

(1) The simplest processing uses zero-order thresholding of the count accumulations in the successive one-second measurement intervals. Thus, the gamma ray count for interval k + 1 is not transmitted unless it varies by more than, say, T units from the previous transmitted value. The disadvantage of such a scheme is that, during periods of rapidly fluctuating counts and for low telemetry rates, the thresholding method may offer almost no data compression. Further, it can require frequent formatting change and can put severe demands on the control of the buffer queue.

(2) The accumulation of histograms for each measurement over some interval, large compared to the one-second counting time, offers several possibilities for data compression. From the histogram, the CDS can readily compute the mean, variance, and other statistical quantities of interest as well as giving the maximum and minimum counts and their relative frequencies and the count having highest frequency (the mode). Furthermore, these statistics can themselves be thresholded so that they are transmitted only when they change by more than some fixed percentage from the previously transmitted values.

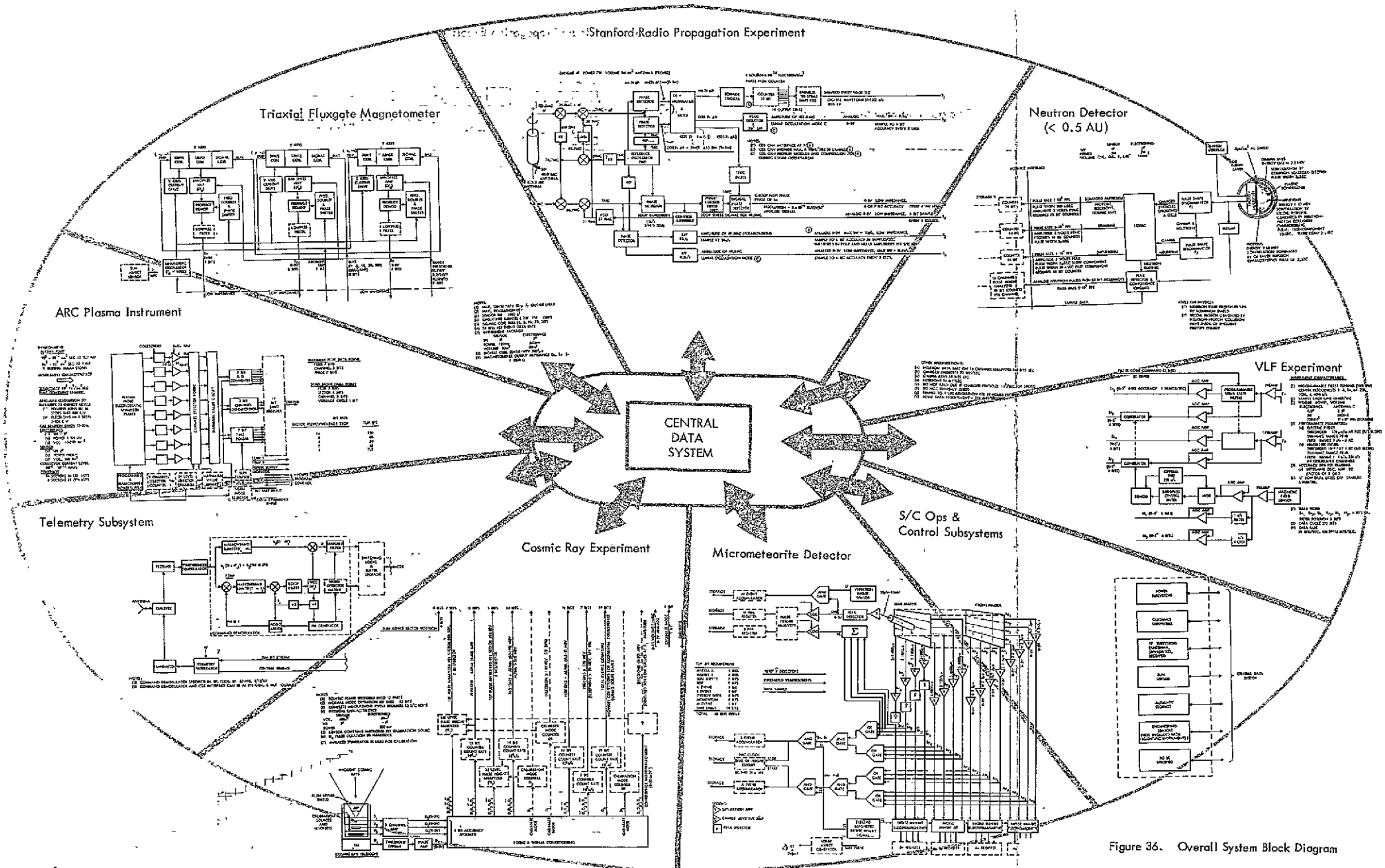


Figure 36. Overall System Block Diagram

Compression of the histogram data can also be achieved by use of sample quantiles. It has been shown that as few as four sample quantiles are sufficient to give efficiencies exceeding 90 percent and 73 percent, respectively, for estimates of the mean and variance from a normal population. The disadvantage of using quantiles is that, for severely nonnormal populations, the estimate of the variance can be substantially in error. Use of more than four quantiles will give improved results, but obviously at the expense of reducing the compression ratio.

8.3.4.2 VLF experiment:-- The instrument schematic is given in Figure 36. The six lines marked E_{x1}, ..., M_2 denote the interface with the CDS. The programmable filter has a bandwidth that is 10 percent of the center frequency. This means that, for the center frequencies of 16 Hz and higher, the sample rate of one sample per second is likely to result in aliasing. For each center frequency, the data output is represented by 39 bits, assuming 6 bits for each of the six lines and 3 bits to identify the center frequency (1 out of 7) used. If the center frequency is always stepped through the same cyclic order, then only one bit is needed to identify the start of the cycle.

Processing alternatives:

(1) Simple thresholding as in paragraph 8.3.4.1, (1).

(2) At low telemetry rates, each measurement can be stored over a number of sample periods and the mean, variance, and other quantities periodically computed and transmitted. These quantities can be thresholded as in paragraph 8.3.4.1,(2).

(3) Each measurement can be sampled at rates greater than one sample per second and digital filtering used to determine the spectral content of the signals. This spectral information can be used either to control the rate at which samples are transmitted (providing the telemetry rate allows for such leeway) or to provide indication to the ground of the data activity. In the former case, the CDS would have the concomitant task of altering the data flow from the other experiments, under some priority schedule, to permit the increased data rate.

(4) Orthogonal polynomials or Fourier series expansions for each measurement over a number of intervals. Such expansions result in best least squares approximations to the data, and a sufficient number of terms can be computed so as to give an error term less than some desired amount. Thus, the possibility exists that M coefficients can be transmitted in place of N data samples, where $M < N$; and, hence compression can be achieved. This method will be suitable for those measurements having lower precision requirements and where the ground station processing is likely to use least squares techniques.

8.3.4.3 Cosmic ray experiment:-- The instrument arrangement is shown in Figure 36. The bit requirement is 67 bits per sector. The sector sampled is advanced each spacecraft revolution. As with the neutron detector, logarithmic counters offer a way of reducing the bit rate.

Processing alternatives: Same as for neutron detector, paragraph 8.3.4.1.

8.3.4.4 Plasma probe:-- Figure 36 showed the schematic arrangement of the instrument and the interface specifications. The instrument is programmed to cycle through a specific measurement pattern; and, as long as this pattern is in force, it is a simple matter to reduce the bit rate with no additional processing. Consider the scan mode data word: 3 bits are used to describe which of 8 channels is selected; 4 bits to give the sector (1 out of 15); 7 bits for the flux measurement; and 1 bit to specify the energy level (analyzer plate voltage) cycle. It is clean that the channel and sector bits are superfluous if the channels are selected in fixed cyclic order and the sectors sampled sequentially. Even at low telemetry rates when some of the sectors are omitted, the ordering is still fixed. Occasionally, it is desirable to transmit the channel and sector information as a check on proper instrument performance. In the maximum mode, channel identification is superfluous because of the fixed order of channel selection. The calibration signals need be sent only if they differ by more than some allowable tolerance from the desired values, a check easily implemented by the CDS. The CDS can also be used to perform the maximum mode functions that are currently shown in Figure 36 as being done within the instrument. As with all other instruments, the easiest way to reduce the average bit rate is simply to restrict the operating period of the instrument.

Processing alternatives:

(1) Simple thresholding as in paragraph 8.3.4.1, (1). In the scan mode, successive samples of the same measurement (i.e., same energy level, channel, and sector) occur once every 384 spacecraft revolutions. Thresholding of measurements for adjacent sectors for the same channel and energy level and for adjacent channels for the same sector and energy level should also be considered as a means of obtaining compression. Similarly, thresholding can be used in the maximum mode.

(2) Histogram compilation and analysis as in paragraph 8.3.4.1, (2), for both the scan mode and maximum mode. A modification of the instrument sampling format might be considered whereby the measurement period could be reduced from the current value of 384 revolutions. This would produce more samples per unit time and would make the histogram analysis more meaningful by reducing the time needed to acquire an appropriately large number of samples. In order to do this, it would be necessary to multiplex all eight channels, in effect, sample them all in each of the 15 sectors.

8.3.4.5 Triaxial fluxgate magnetometer:-- The basic instrument schematic and interface is given in Figure 36. A realistic approach to data processing requires that the low-pass filter for each axis have a cutoff frequency sufficiently high to pass all desired information. For all three gates combined, a single data sample specifies either implicitly or explicitly 32 bits: 8 bits for each flux measurement; 2 bits for the sun aspect sensor position; 2 bits for the dynamic range position; 3 bits for the dc offset position; and 1 bit for the fluxgate physical orientation. This assumes that sampling is done four times per revolution. Here, as with the plasma probe, some savings can be made if the same sampling arrangement is always used. However, it may be desirable to sample more frequently in order to determine the spectral characteristics of the sensor outputs. This spectral information can be used to determine an appropriate sampling rate or to govern the rate at which measured samples are transmitted. All processing, including spectral analysis and compensation for the spin effect of the spacecraft on the magnetic field measurements, can be performed digitally.

Processing alternatives:

(1) Simple thresholding as in paragraph 8.3.4.1, (1).

(2) Histogram compilation and analysis as in paragraph 8.3.4.1, (2).

(3) Spectral analysis of sensor outputs. If the analysis indicates that signal frequency content is higher than can be handled by the available telemetry rate, then some points of the spectral distribution can be transmitted.

8.3.4.6 Stanford radio propagation experiment:-- Figure 36 shows the schematic arrangement of the instrument and notes the appropriate interface points. The most substantial bit requirement occurs on line S_5. We can assume that the CDS determines the maximum and minimum values (6-bit precision) of S_5 once per revolution from 36 or 128 samples. Line S_1 accumulates counts; and, therefore, histogram methods are applicable.

Processing alternatives:

(1) Simple thresholding as in paragraph 8.3.4.1, (1) -- applicable to all lines.

(2) Histogram analysis as in paragraph 8.3.4.1, (2) -- applicable to all lines, particularly S_1.

(3) Orthogonal polynomials as in paragraph 8.3.4.2, (4) -- applicable to all lines except S_1.

8.3.4.7 Conclusions:-- Several points are clear as regards effective use of the CDS in handling the variety of signals generated by the instruments. In general, it seems better to employ averaging methods (e.g., computation of mean, variance, and spectral distribution) rather than omit data samples when the data bandwidth exceeds the available telemetry rate. In this way, the CDS is being used to effect compression; and aliasing errors due to insufficient sampling rates are minimized.

It is also clear that a variety of algorithms could be stored by the CDS so that, by monitoring the data from each instrument, the appropriate algorithm can be selected. This "tailoring" of the processing to each channel is a distinct advantage possessed by a digital computer.

8.4 Addition of Magnetic Tape Storage

The flexibility of a stored program central data system such as MULTIPAC allows the addition of a mass memory unit, such as magnetic tape, to the system for storate of output data at times the telemetry link is not in use. The program technique used for this type of unit will depend entirely on the sophistication of this unit. A very unsophisticated unit could be used which simply stores data on a long loop of tape to be transmitted later. For such a unit, the programmer can consider it another telemetry unit. It is strongly recommended that the unit be interfaced with a module very similar to the command/telemetry unit. In addition to transmitting data to this unit, the program need only start and stop the unit. The start and stop commands will be given very seldom, and the data will be sent to the unit at the slow telemetry rate.

A more powerful implementation would be to design the tape unit to run at higher speeds and put the data on the tape in blocks separately by record gaps. In this case the formatting and outputting routine will be more efficient since a block of words can be outputted each time the routine is entered. A simple subroutine can be used to time out the record gaps for start and stop. Later when the telemetry link is active, MULTIPAC can read a block of data at a time into memory and output to the telemetry at the telemetry rate.

If, in addition, the tape has the capability of fast forward and rewind, programs which are seldom used, such as diagnostics or backup programs, can be stored on the tape. This will result in a great deal more diagnostic capability while the spacecraft is out of contact with the Earth.

9.0 REPROGRAMMING AROUND FAILURES

The reliability of the MULTIPAC system is achieved through its ability to reprogram around failures. This section describes some of the techniques used to accomplish this reprogramming.

It does not seem feasible at this time to fly enough memory to perform all the diagnostics and automatic reprogramming. A more realistic approach is to diagnose the error through the command and telemetry links; to reassemble the program on a ground-based computer; and then transmit the new program to the spacecraft. In general, it will not be necessary to reprogram all of the memory.

A typical system will have three logic units and enough memories to have three complete processors operating simultaneously. A processor is defined here to mean enough programmable units to program one or more of the CDS tasks. The most likely division of work into these three processors will be: one responsible for inputting and outputting data; another responsible for telemetry buffering; and the third responsible for data reduction processing.

The discussion will be divided into failures of various units (e.g., registers, logic units, etc.).

9.1 Complete Failure of a Register

Most failures in a register will cause complete register failure. Data is moved in and out of the register serially. A failure in a flip-flop, input gating or output gating will cause all bits of the word to fail.

The major consequence of a register failure is the loss of the 12 input lines and 12 output lines. If these lines are not redundantly connected to another register, then this represents a permanent loss of data. In general, it is expected that enough resisters exist so that science lines can be redundantly connected to two registers. Thus, the loss of a single register will not cause the loss of any science data.

In the case of those lines connected redundantly to another register, the I/O tables in the I/O processor are simply changed to reflect the alternate register for the connection of the devices concerned. This programming task could be included in the I/O processing routines so that only a simple command to update these tables is necessary. However, since it takes very little time to update the tables directly from the command link, this latter approach is preferable to keep the CDS programs as simple as possible.

In addition to use as I/O interfacing, the registers are used as scratch and indexing by programs. The ground-based computer should keep a table of register usage for each of the programs and should reassemble those programs using a failed register. As long as the number of failures is small, there should be spare registers for this purpose. If this is not the case, then the programs will have to be reassembled using fewer registers. Frequently, there exists a reprogramming solution to accomplish a process without relying so heavily upon available registers. Keeping process parameters in registers is usually the most efficient in time. Most often a loop which keeps constants in registers can be reoriented to retrieve from program (not data) memory each time they are needed at a slight loss in processing speed.

9.2 Complete Failure of a Logic Unit

Any failure of a logic unit is likely to cause associative failure (due to failed logic), and some diagnostics should be attempted on a logic unit to determine to what use, if any, this unit can be assigned. In this discussion we will consider only the complete failure of the logic unit.

The failure of a single logic unit out of the three will primarily affect throughput. Two logic units should be able to supply more than the minimum required processing load of the central data system. It is the extra tasks, such as data reduction, which will be affected. Since initially all logic units will be working, it is reasonable to assume that as many data reduction tasks as possible will be added to the CDS program requirements to use up the MULTIPAC processing capability. In the event of a logic unit failure, a cutback would then be made in the amount of data reduction, particularly at high telemetry rates. At low telemetry rates it is likely that no degradation in performance will occur.

This reduction to two logic units will require reprogramming the entire mission. Thus, it is imperative that this be done ahead of time in case such a failure should occur.

9.3 Memory Failures

There are very few single failures in a memory which will cause a complete memory failure. However, multiple failures particularly three or more, will tend to make it very difficult to use the memory. For example, programming around loss of every other word and one of the middle bits in every word may be more trouble than it is worth. In this case one would consider the memory totally failed. Complete failures and some of the more likely partial failures will be discussed in following paragraphs.

9.3.1 Complete failure.-- If more than six memory units are initially available, there will be very little overall effect, except for those data reduction algorithms which need large data stores. The computer on the ground will have to go over all programs which reference this memory and reallocate the storage. If this memory contains programs, these programs will now have to be read into another memory and the program memory paging changed.

If the failure brings the total number of memories below two per logic unit, then the processing throughput will decrease. When the programs and data are in the same memory, an extra memory cycle time is required on all instructions referencing memory. This extra time is due to the loss of data fetch and instruction fetch overlapping which results from using two memories. Assuming equal memory reference and register reference instructions, this will decrease the speed for this one process by one-third.

Enough memory failures will have to occur to bring the CDS below four operating memories before memory failures will prevent operating two processors simultaneously. However, once the system is reduced to four working memories, the amount of data reduction processing capability will be seriously limited since most of these algorithms tend to use large amounts of memory space.

9.3.2 Partial failures.--

9.3.2.1 Complete loss of the memory register:-- If this register section is completely failed, then the memory unit is completely failed.

9.3.2.2 Partial loss of addressable words:-- This can occur due to bits of the memory address register failing or an x or y decoder failing. If conveniently addressable segments remain, such as halves or fourths of memory, then the unit really behaves as a smaller memory. If the useful words are scattered throughout memory, then the memory can only be used as random tempory storage or for constants. In this latter case, its effect is very similar to the complete memory failures described above.

9.3.2.3 Loss of a bit:-- The memory cannot be used for programs since there is no way to mask the effect. The use as data memory is limited, especially if the failure is intermittent or in the low order bits.

If the failure is in one of the high order bits, the unit could be used to store small data with some extra programming to mask off this bit. If the bit fails to a zero, only mask negative numbers are needed and vice versa for a failure to a one. If the failure is in the low order bits, it is necessary to shift the data word on every access to memory, which probably precludes widespread use.

9.3.2.4 Loss of a single word:-- This is a trivial problem for either program or data storage and can be easily taken care of by the assembler in the ground-based computer.

9.4 Command Override Procedure

The uplink commands are 16 bits in length. The last four bits (the four most significant bits) are used to distinguish between normal commands and command override commands. When these four bits are all 0, a normal command is assumed. The other 15 combinations of these four bits are used for command override. Each of these 15 address 15 different locations in the MULTIPAC system: three to each of 5 logic units. When less than 5 logic units are in the MULTIPAC system, the unused commands will act like command override without performing any command override function. In other words, these unused commands will be treated like override commands, but will be sent to nowhere.

The three addresses within a logic unit addressed by the command override feature are the two accumulators and the instruction shift register. A command sent to the instruction shift register will override any other instruction entering the instruction shift register and this new instruction will be performed as if it came from the program memory.

When overtaking the MULTIPAC system, the first procedure, normally, is to turn off all logic units. A logic unit is turned off with an override command instruction to set the program memory page to "0". Since there is no program memory whose address is "0", this will effectively send zeros continuously to the instruction shift register. Zeros are treated as NOP instructions by the MULTIPAC logic unit.

The procedure to turn off all logic units is to send the instruction SPMP (set program memory page) "0" to each of the logic units in turn. SPMP "0" will set the program memory page of each logic unit to "0". After all logic units have been disabled, the procedure is either to enable some program stored in a known good memory or else to bootstrap in a program from the ground into a memory.

To start the program at some program memory N at address A, first the command to load Accumulator 1 with the Address A is sent. The instructions JMPR ACC1 are then sent to the instruction shift register. This jump through register instruction will set the program counter to the address in Accumulator 1. Since the program memory page is still "0", the program counter will not increment and the address A will remain in the program counter until the program memory page changes. Next, the instruction SPMP N is sent to the instruction shift register, and when this instruction is performed, the program memory page will switch to the requested memory and the instructions will begin to be performed from the address stored in the program counter.

To bootstrap in a program into memory, the data memory switch of a logic unit is set to the proper memory with a SDMP instruction and then the following three commands can be used to load each word into memory. First, the address is loaded into Accumulator 1, and second, the data is sent to Accumulator 2. The third command is the instruction STA2 indexed by Accumulator 1. Commands from command override portion of the second unit are sent to the instruction shift register as one word followed by many words of all zeros. Thus, if a first word of a two-word instruction is sent to the instruction shift register, the second word will be all zeros. The STA2 indexed by Accumulator 1 instruction will have an address of "0" indexed by Accumulator 1, or, in other words, the address will be that of Accumulator 1. This will send Accumulator 2 to this address in memory. This procedure can be repeated over and over again until enough words are scored in memory to load programs with normal command words. Befor this loading program is entered, the diagnostic procedures of the next section should be followed to determine what modules are working properly.

9.5 Reprogramming Methods

Reprogramming may be accomplished from the ground by simply sending up a new section of code. Given any failure, the next most efficient program usually involves a complete reorganization. This entails approximately 1000 words being sent up. At 10 bits per second, this requires less than one-half hour to transmit. Total reprogramming (~6000 words) would take only 3 hours to accomplish. On the other hand, reprogramming from within simply cannot cope with this kind of reorganization. Specifically, any reprogramming requires changing code, hence a reassembly.

A relabeling process is possible for register failures only. Any failure other than a register will require a total reorganization of some routine to prevent inefficiency of running time and memory storage utilization. There seems to be no real use for reprogramming which can be accomplished on-board so that the command link transmission speed is the factor which determines the amount of reprogramming done in a given time.

A very critical area is the problem of determining "what" has failed whenever it becomes apparent that "something" has failed. A great deal of the failure detection is going to occur on the ground. It does not seem practical to put sophisticated detection programs on-board, since these generally take a great amount of running time and memory. During the period when the spacecraft is not transmitting to ground, a reasonable amount of failure detecting can be run, but this does not detect failures that occur during the transmission period.

There seems to be no useful diagnostics which could be run on-board, since these are far more complex than failure detection. Consequently, the discussion in this section is concerned with how to diagnose from the ground once the failure is apparent on the ground. In most cases all activity will have to be stopped and all modules cleared so that they will not conflict with the diagnosing process.

9.5.1 Diagnostic tests.-- These diagnostics should not be very extensive. Rather, they should be a short sequence of tests which are optimal for the length of the sequence. The main purpose of these diagnostics is to get some confidence quickly in most of the units which are not definitely known to be bad. If these tests fail to reveal trouble, then some more extensive tests may be run. There is a reasonable chance that the simple echo and register tests could be included in every memory unit. The remaining tests will probably be too large. However, they might be stored on a tape. The time estimates involve transmitting everything from the ground, but tape could be used for such things. It is encouraging that even the worst case of numerous duplicate transmissions from the ground is not exorbitant in time. As units fail, the search is reduced because the unit no longer has to be considered. Looking at the problem in this light, it seems clear that the bootstrap test can always be done in quite reasonable time unless a large number of failures occur at once.

9.5.1.1 A simple echo:-- The simplest possible program which generates feedback is necessary. This would probably consist of a sequence of words being sent back via telemetry, for example,

```
TEST:    LDA1     WORD1
         OUT      TEL1
         INP      MASK,RFLAG
         SKP1     NEZ
         JMP      .-3
         LDA1     WORD2
          .
          .
          .
          .
         JMP      TEST
```

where the code is repeated for each word to be sent back. In this program the word to be tested is read into Accumulator 1 and is outputted to a

telemetry register. The flag is then inputted from that telemetry register, which is tied to some input interface channel as specified by MASK and RFLAG. This flag is loop tested until another word is needed and then the processing is continued. This program uses one memory, one logic unit and one telemetry register. If the desired sequence comes back some specified number of times, then a basic processor has been located. Otherwise, the program must be reassembled and retransmitted using some other combination of the three units. This can be done in an optimal manner with some analysis.

A program such as the simple echo routine should include a frame synchronization (FS) code. This would assure at least one word having the high correlation properties of a Barker code and thereby form a very simple frame format to make the ground synchronization and decommutation problem easier. One word could be echoed which contains a 7-bit Barker code plus 5 bits of data.

9.5.1.2 Register test:-- The next step is check out all registers A basic processor exists but cannot be very useful unless a reasonable number of registers are alive. In addition, checking registers is at least logically simple. The program would be very similar to that above except that the words to be transmitted would be first copied into a register, then into the telemetry. This is a very basic test and can test all registers with one simple program.

9.5.1.3 Creating a full processor:-- The next two steps are to perform an instruction test for all the modules used up to now and then to search for a good data memory. A full processor needs two memories, a logic unit, and probably one to four registers. The search for a good data memory can be performed on-board, since we have already determined through the instruction test that the selected modules work properly. This program should interact often with the ground so that unforeseen problems may be detected. At this time it will be more efficient to test all memories rather than just to look for a good one.

9.5.1.4 Test of remaining logic units:-- When the above is completed, the system is known to have at least one full processor plus a number of available memories and registers. The next test should be the complete checkout of the other logic units. This can be done completely on-board by the good processor monitoring the output of a standard program residing in a good memory. As soon as one unit checks out, it is turned off and the other unit is set to execute from the same test program (same memory unit).

9.5.1.5 I/O test:-- The final test is to check out all the I/O devices and generate a device number-to-register map for use by the I/O programs. Absolutely failed units should be reported to ground, although reprogramming would not usually be required. The other telemetry unit should be tested at this time if not tested previously.

9.5.2 Timing.-- Overall time for a complete set of diagnostics where everything is transmitted from the ground is on the order of 1-2 hours. This assumes an unlikely 2000 words of transmission and one hour of analysis. Running diagnostics from an on-board tape unit might appreciably reduce this time by reducing transmission time and allowing more lengthy self-checking routines to be run. Thus, it looks like the bad failure situation might take two hours of diagnosis and three hours of reprogramming. If very definitive diagnostics are desired, then the time increases, simply because they must be run for long periods of time. It is clear that intermittent failures will cause either long periods of diagnosis or living with intermittently bad data.

9.6 Ground Software

Ground-based software must emphasize the ability to diagnose and reprogram in the shortest possible time. This primarily implies a large collection of preassembled routines using all the combinations of available hardware. This is, or course, impractical. What really is needed is: 1) a diagnostic generator which uses failure history to reduce its output, and 2) various organizations determined by sets of available units of the running software which are abstract in the actual units utilized. This latter means assuming some subset of units being available and writing the most efficient code for the situation. Parameters to each such encoding would be the actual unit numbers (switch addresses). This ability implies an assembler of only moderate complexity with relatively simple macro features.

No time should be wasted on any kind of compiler. Code simply must be as efficient as possible, which means only machine language coding.

A computer must be available at all times for reassembling programs. Some on-board problems, such as failures, will be solved only by having a programmer generate more code in real time. If possible, the computer should take care of transmission to the on-board system. Extremely desirable would be a diagnostic generator (as above) which checks out the system automatically when needed. There is probably no need of human analysis most of the time. This is more true of diagnostics than reprogramming.

10.0 CONCLUSIONS AND FUTURE RECOMMENDATIONS

This contract began as a study of data formatting and data system organizations for lightweight deep space probes. The first year of this study, which has been reported previously, recommended that the central data system use stored program computer concepts, that data formatting should be very flexible, and that data reduction algorithms should be used whenever possible. The initial contract was extended to develop the multiple pooled central data system concept described in this report.

MULTIPAC is a central data system using stored program computer concepts which would give future spacecraft extensive data processing capability for variable data formatting, sampling and converting analog information from experiments, and performing data reduction on experimental data to improve information transfer on a limited telemetry bandwidth. An organization consisting of pools of modules organized by the program is used to achieve an extremely high reliability for extended deep space probes. In the event of a failure, this multiple pooled organization allows reprogramming around the failed modules, permitting the surviving modules to be utilized optimally.

In addition to the ability to recover from failures, this multiple pool organization replaces the current technique of designing a new data system for each probe, with a standard "off the shelf" central data system, which is programmed by software to perform as a flexible data management system. One typical organization was used as an example throughout this report. This typical MULTIPAC configuration, a 16-watt system, including 12,288 words of memory, can handle about 200 science and engineering input lines and 200 output lines. This typical system could simultaneously schedule sampling of the experiments, perform needed analog-to-digital conversions, reduce data using histograms or other data reduction techniques, and then format the data for transmission by the telemetry subsystem. Since a computer organization is used, wide variations in formatting, sampling schedule, and other data management tasks are easily accommodated. These changes can be made later in flight from the ground after the data has been analyzed. It is at this time that data reduction techniques are quite powerful, since after the flight has been in progress for some time, enough may be known about the data to effectively perform data reductions on the raw data. In addition, if an experiment has failed, the part of the data format transmitted to earth from that failed experiment can be used by other experiments.

The above system has an extremely high probability of surviving 36 months (the longest mission considered). However, with the very pessimistic failure rate of 10^{-5} LSI circuits per LSIC-hour (1 percent per 1000 hours), the probability of system survival is 0.0001. For more realistic failure rates of 10^{-6} or 10^{-7}, the corresponding figures are 0.92 and 0.999, respectively. These failure rates are for a minimum operable configuration of one processor, 2048 words of memory and 83 percent of

the input-output interface lines working properly. This configuration is more than enough to perform scheduling and sampling of the science and engineering lines and data formatting which is the capability of present-day fixed format central data systems.

The present design is expandable to five processors and 32,768 words of storage. Each of these processors will act as a computer with an instruction rate of 15 microseconds. These limits are arbitrary and simple changes to the system design can be made if greater memory storage and/or computers are needed. This fully expanded MULTIPAC system will require 32 watts of power, and will handle an extensive input-output interface to the experiments, many times greater than that of present day space probes. The generality of the design is such that it can easily handle input/output devices not included in this design with existing modules or with the addition of new modules. These new modules are easy to interface, require very few interconnections, and may be directly addressable by the programs.

As this design moves into hardware implementation, it is probable that some changes will be made due to further analyses of the system's requirements. Before final implementation, it is recommended that a typical mission be programmed, and that diagnostics be written to determine whether they ought to be transmitted from the ground (the most likely), or stored in memory. These programming tasks may result in recommendations for some changes in the overall design. It is expected that such design changes will be limited to change of instruction repertoire, in which case only the design of the logic unit need be affected. The register and memory can remain exactly the same, and the LSIC's of these modules may be released before programming is done. Programming a typical mission will give a closer estimate of memory requirements and the amount of data reduction processing capability available to the experimenter.

These programming tasks may be accomplished while a breadboard is being built. Breadboarding costs about the same (assuming integrated circuits are used in place of the LSI circuits) as performing a computer simulation and is a far more accurate representation of the final system.

Without a mass storage device aboard the spacecraft, reprogramming from the ground will require a command link capability of at least 10 bits per second. Even at that rate it is possible that a failure could put a spacecraft out of contact with earth for one to two hours. If this is an unacceptable delay, it may be desirable to add a simple commutator under control of the command decoder which will bypass the central data system. This bypass, which would be used while reprogramming, could simply transmit the raw data with frame syncs and parity in a fixed sampling sequence.

In conclusion, a very low power, extremely flexible central data system has been described which can be reprogrammed from the ground to either change its characteristics or to program around failed components. This design can be used for all (or most) future deep space probes replacing the present data systems which are specifically designed for each flight.

REFERENCES

1. A Study to Determine an Efficient Data Format and Data System for a Lightweight, Deep Space Probe. NASA CR-73211, Contract No. NAS2-3255; February 1968.

2. MULTIPAC, A Multiple Pool Processor and Computer for a Spacecraft Central Data System. Research Report No. NASA CR-73262; March 1969.

3. Cricchi, J.R; Lancaster, E; and Strull, G; A Large-Scale Complementary MOS Memory. Supplement to IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-3, No. 6; November 1967.

APPENDIX A

RELIABILITY PROGRAM

The results tabulated in Section 6.0 using the general model of Figure 31 of that section, was written on a time-sharing terminal using a language called TELCOMP. This is typical JOSS language (simplified ALGOL) similar to CAL, a more generally known language. The program is shown in Table A1 and its output upon the command "DO PART 6" is shown in Table A2.

Abbreviations used:

- CG - Clock Generator (oscillator + squaring)
- TC - Timing Counter
- TC2OF3 - 2 of 3 TC's
- LU - Logic Unit
- M - Memory Unit
- MS - Memory Storage Element
- R - Register
- TM - Telemetry Unit
- CM - Command Unit
- RP - Register Pair (redundantly connected I/O)
- DA - D/A Register
- RATYP - Typical Analog Comparator Return Reliability (must have one D/A and corresponding I/O register)
- RADGR - Typical Digital Input Reliability
- L1TYP, L2TYP - Partial calculations of RATYP
- L1DGR, L2DGR - Partial calculations of RADGR

TABLE A1

PROGRAM FOR RELIABILITY

```
1.01 DO PART 2 FOR MNR=1 FOR N=1 FOR PC=.2
1.02 CG=PS
1.03 DO PART 2 FOR MNR=2 FOR N=3 FOR PC=1
1.04 TC2OF3=PS
1.05 DO PART 2 FOR MNR=3 FOR N=3 FOR PC=14
1.06 LU3OF3=PS
1.07 DO PART 2 FOR MNR=1 FOR N=3 FOR PC=14
1.08 LU1OF3=PS
1.09 DO PART 2 FOR MNR=6 FOR N=6 FOR PC=8
1.10 M6OF6=PS
1.11 DO PART 2 FOR MNR=1 FOR N=6 FOR PC=8
1.12 M1OF6=PS
1.13 DO PART 2 FOR MNR=1 FOR N=2 FOR PC=3
1.131 RP=PS
1.15 CM1OF2=PS
1.16 DO PART 2 FOR MNR=1 FOR N=2 FOR PC=4
1.17 TM1OF2=PS
1.18 DO PART 2 FOR MNR=6 FOR N=6 FOR PC=3
1.19 R6OF6=PS
1.20 DO PART 2 FOR MNR=5 FOR N=6 FOR PC=3
1.21 R5OF6=PS
1.22 DO PART 2.04 FOR MNR=6 FOR MR=RP FOR N=6
1.23 RP6OF6=PS
1.24 DO PART 2.04 FOR MNR=5 FOR MR=RP FOR N=6
1.25 RP5OF6=PS
1.26 DO PART 2 FOR MNR=1 FOR N=1 FOR PC=4
1.27 DA=PS
1.28 DO PART 2 FOR MNR=2 FOR N=3 FOR PC=14
1.29 LU2OF3=PS
1.30 DO PART 2 FOR MNR=4 FOR N=6 FOR PC=8
1.31 M4OF6=PS
1.32 DO PART 2 FOR MNR=2 FOR N=6 FOR PC=8
1.33 M2OF6=PS
1.34 DO PART 2 FOR MNR=2 FOR N=6 FOR PC=136
1.35 MS2OF6=PS
1.36 DO PART 2 FOR MNR=4 FOR N=6 FOR PC=136
1.37 MS4OF6=PS
1.38 DO PART 2 FOR MNR=6 FOR N=6 FOR PC=136
1.39 MS6OF6=PS
1.40 DO PART 2 FOR MNR=1 FOR N=6 FOR PC=136
1.41 MS1OF6=PS
1.60 TYPE #,#,#
1.70 TYPE MO,FR IN FORM 6
1.71 TYPE #,CG,TC2OF3,LU3OF3,LU2OF3,LU1OF3,M6OF6,M4OF6,M2OF6,M1OF6
1.715 TYPE MS6OF6,MS4OF6,MS2OF6,MS1OF6
1.72 TYPE TM1OF2,CM1OF2,R6OF6,R5OF6,RP,RP5OF6,RP6OF6,DA,#
 1.91 DO PART 4
1.92 DO PART 5

2.03 MR=EXP(-PC*FR*730*MO)
2.04 NS=N,PS=0
2.05 SET NF=N-NS
2.06 DO PART 3 FOR J=N,NS,NF
2.07 SET PRB=FCT[N]/(FCT[NS]*FCT[NF])*MR↑NS*(1-MR)↑NF
 2.09 PS=PS+PRB
2.10 NS=NS-1
2.11 TO STEP 2.05 IF NS>=MNR
```

TABLE A1.-- Continued

PROGRAM FOR RELIABILITY

```
3.0 SET FCT[J]=1
3.1 FCT[J]=FCT[J]*I  FOR I=1:1:J

4.01 L1TYP=DA*(1-DA)*R6OF6
4.02 L2TYP=DA↑2*RP6OF6
4.03 RATYP=2*L1TYP+L2TYP
4.04 L1DGR=DA*(1-DA)*R5OF6
4.05 L2DGR=DA↑2*RP5OF6
4.06 RADGR=2*L1DGR+L2DGR
4.81 TYPE L1TYP,L2TYP,RATYP,L1DGR,L2DGR,RADGR,#,#

5.01 REL3X=CG*TC2OF3*LU3OF3*TM1OF2*CM1OF2
5.02 REL3=REL3X*M6OF6*RP6OF6*RATYP
5.03 REL3A=REL3X*RP5OF6*RADGR*M6OF6
5.04 REL3M=REL3X*MS6OF6*RP6OF6*RATYP
5.05 REL3MA=REL3X*MS6OF6*RP5OF6*RADGR
5.06 REL2X=CG*TC2OF3*LU2OF3*TM1OF2*CM1OF2
5.07 REL2=REL2X*M4OF6*RP6OF6*RATYP
5.08 REL2A=REL2X*M4OF6*RP5OF6*RADGR
5.09 REL2M=REL2X*MS4OF6*RP6OF6*RATYP
5.10 REL2MA=REL2X*MS4OF6*RP5OF6*RADGR
 5.11 REL1X=CG*TC2OF3*LU1OF3*TM1OF2*CM1OF2
5.12 REL1=REL1X*M2OF6*RP6OF6*RATYP
5.13 REL1A=REL1X*M2OF6*RP5OF6*RADGR
5.14 REL1M=REL1X*MS2OF6*RP6OF6*RATYP
5.15 REL1MA=REL1X*MS2OF6*RP5OF6*RADGR
 5.16 RELOX=CG*TC2OF3*LU1OF3*TM1OF2*CM1OF2
5.17 RELO=RELOX*M1OF6*RP6OF6*RATYP
5.18 RELOA=RELOX*M1OF6*RP5OF6*RADGR
5.19 RELOM=RELOX*MS1OF6*RP6OF6*RATYP
 5.20 RELOMA=RELOX*MS1OF6*RP5OF6*RADGR
5.83 TYPE FORM 1
5.84 LINE
5.85 TYPE REL3,REL2,REL1,RELO IN FORM 2
5.86 LINE
5.87 TYPE REL3A, REL2A,REL1A,RELOA IN FORM 3
5.88 LINE
5.89 TYPE REL3M,REL2M,REL1M,RELOM IN FORM 4
5.90 LINE
5.91 TYPE REL3MA,REL2MA,REL1MA,RELOMA IN FORM 5
 5.92 TYPE #,#

6.0 DO PART 1 FOR MO=12:12:36 FOR FR=10↑-7
6.1 DO PART 1 FOR MO=12:12:36 FOR FR=10↑-6
6.2 DO PART 1 FOR MO=12:12:36 FOR FR=10↑-5

FORM        1
                    REL: 3LU,6M        2LU,4M        1LU,2M        1LU,1M
 FORM        2
   FULL I/O             .####         .####         .####         .####
 FORM        3
   83% I/O              .####         .####         .####         .####
 FORM        4
   FULL I/O (W MS)      .####↑↑↑      .####↑↑↑      .####↑↑↑      .####↑↑↑
FORM        5
   83% I/O (W MS)       .####↑↑↑      .####↑↑↑      .####↑↑↑      .####↑↑↑
FORM        6
      SOLAR PROBE ## MONTH RELIABILITY FOR FAILURE RATE= #.#↑↑↑
```

TABLE A2

OUTPUT OF PROGRAM

SOLAR PROBE 12 MONTH RELIABILITY FOR FAILURE RATE= 1.0-07

CG=	.999824815
TC2OF3=	.999997701
LU3OF3=	.963876601
LU2OF3=	.999557899
LU1OF3=	.999998189
M6OF6=	.958823756
M4OF6=	.999993295
M2OF6=	1
M1OF6=	1
MS6OF6=	.489282131
MS4OF6=	.978202277
MS2OF6=	.99990281
MS1OF6=	.999997993
IM1OF2=	.999987765
CM1OF2=	.999993112
R6OF6=	.984355664
R5OF6=	.999897397
RP=	.999993112
RP5OF6=	.999999999
RP6OF6=	.999958671
DA=	.996502132
L1TYP=	3.4311027*10↑ -3
L2TYP=	.992975458
RATYP=	.999837664
L1DGR=	3.48527548*10↑ -3
L2DGR=	.993016498
RADGR=	.999987049

REL:	3LU,6M	2LU,4M	1LU,2M	1LU,1M
FULL I/O	.9238	.9992	.9996	.9996
83% I/O	.9240	.9993	.9998	.9998
FULL I/O (W MS)	.4714+00	.9774+00	.9995+00	.9996+00
83% I/O (W MS)	.4715+00	.9776+00	.9997+00	.9998+00

TABLE A2.-- Continued

OUTPUT OF PROGRAM

SOLAR PROBE 24 MONTH RELIABILITY FOR FAILURE RATE= 1.0-07

CG=	.999649661
TC2OF3=	.999990818
LU3OF3=	.929058102
LU2OF3=	.998267224
LU1OF3=	.999985775
M6OF6=	.919342995
M4OF6=	.999947746
M2OF6=	.999999997
M1OF6=	1
MS6OF6=	.239397004
MS4OF6=	.885811874
MS2OF6=	.997883975
MS1OF6=	.999909185
TM1OF2=	.999951231
CM1OF2=	.999972519
R6OF6=	.968956074
R5OF6=	.999593517
RP=	.999972519
RP5OF6=	.999999989
RP6OF6=	.999835126
DA=	.993016499
L1TYP=	6.71945054*10↑ -3
L2TYP=	.985919188
RATYP=	.999358089
L1DGR=	6.931913*10↑ -3
L2DGR=	.986081756
RADGR=	.999945582

REL:	3LU,6M	2LU,4M	1LU,2M	1LU,1M
FULL I/O	.8531	.9970	.9987	.9987
83% I/O	.8537	.9977	.9995	.9995
FULL I/O (W MS)	.2221+00	.8832+00	.9966+00	.9987+00
83% I/O (W MS)	.2223+00	.8838+00	.9974+00	.9994+00

TABLE A2.-- Continued

OUTPUT OF PROGRAM

SOLAR PROBE 36 MONTH RELIABILITY FOR FAILURE RATE= 1.0-07

CG=	.999474538
TC2OF3=	.999979371
LU3OF3=	.895497365
LU2OF3=	.996179575
LU1OF3=	.999952863
M6OF6=	.881487903
M4OF6=	.999828195
M2OF6=	.999999977
M1OF6=	1
MS6OF6=	.117132676
MS4OF6=	.743359332
MS2OF6=	.98897748
MS1OF6=	.999263489
TM1OF2=	.999890652
CM1OF2=	.99993833
R6OF6=	.953797399
R5OF6=	.999094156
RP=	.99993833
RP5OF6=	.999999943
RP6OF6=	.999630039
DA=	.989543058
L1TYP=	9.86950861*10↑ -3
L2TYP=	.978833199
RATYP=	.998572217
L1DGR=	.0103382211
L2DGR=	.979195408
RADGR=	.99987185

REL:	3LU,6M	2LU,4M	1LU,2M	1LU,1M
FULL I/O	.7874	.9935	.9974	.9974
83% I/O	.7887	.9952	.9991	.9991
FULL I/O (W MS)	.1046+00	.7387+00	.9864+00	.9967+00
83% I/O (W MS)	.1048+00	.7399+00	.9881+00	.9984+00

TABLE A2.-- Continued

OUTPUT OF PROGRAM

SOLAR PROBE 12 MONTH RELIABILITY FOR FAILURE RATE= 1.0-06

CG =	.998249534
TC2OF3 =	.99977312
LU3OF3 =	.692172553
LU2OF3 =	.963111128
LU1OF3 =	.998462483
M6OF6 =	.656731513
M4OF6 =	.99469357
M2OF6 =	.99999196
M1OF6 =	.999999904
MS6OF6 =	7.8630956*10↑ -4
MS4OF6 =	.0735338386
MS2OF6 =	.588015717
MS1OF6 =	.88613902 4
TM1OF2 =	.998814354
CM1OF2 =	.999327236
R6OF6 =	.854123057
R5OF6 =	.990586456
RP =	.999327236
RP5OF6 =	.999993223
RP6OF6 =	.995970201
DA =	.965566793
L1TYP =	.0283975088
L2TYP =	.928562173
RATYP =	.98535719
L1DGR =	.0329345841
L2DGR =	.932312913
RADGR =	.998182081

REL:	3LU,6M	2LU,4M	1LU,2M	1LU,1M
FULL I/O	.4444	.9366	.9761	.9761
83% I/O	.4520	.9526	.9928	.9928
FULL I/O (W MS)	.5321-03	.6924-01	.5740+00	.8650+00
83% I/O (W MS)	.5412-03	.7042-01	.5838+00	.8798+00

TABLE A2.-- Continued

OUTPUT OF PROGRAM

SOLAR PROBE 24 MONTH RELIABILITY FOR FAILURE RATE= 1.0-06

CG =	.996502132
TC2OF3 =	.999105595
LU3OF3 =	.479102843
LU2OF3 =	.878644571
LU1OF3 =	.98970882
M6OF6 =	.43129628
M4OF6 =	.967100241
M2OF6 =	.99979547
M1OF6 =	.999994996
MS6OF6 =	6.18282726*10↑ -7
MS4OF6 =	9.34050436*10↑ -4
MS2OF6 =	.0994429334
MS1OF6 =	.440685136
TM1OF2 =	.995419313
CM1OF2 =	.997378293
R6OF6 =	.729526197
R5OF6 =	.965742979
RP =	.997378293
RP5OF6 =	.999897618
RP6OF6 =	.984372497
DA =	.932319231
L1TYP =	.046033163
L2TYP =	.855635425
RATYP =	.947701751
L1DGR =	.0609384613
L2DGR =	.869130157
RADGR =	.99100708

REL:	3LU,6M	2LU,4M	1LU,2M	1LU,1M
FULL I/O	.1905	.7836	.9124	.9126
83% I/O	.2024	.8323	.9692	.9694
FULL I/O (W MS)	.2732-06	.7568-03	.9075-01	.4022+00
83% I/O (W MS)	.2901-06	.8038-03	.9640-01	.4272+00

TABLE A2.-- Continued

OUTPUT OF PROGRAM

SOLAR PROBE 36 MONTH RELIABILITY FOR FAILURE RATE= 1.0-06

CG =	.994757789
TC2OF3 =	.99801 6609
LU3OF3 =	.331621838
LU2OF3 =	.774064853
LU1 OF3 =	.970830968
M6OF6 =	.283245858
M4OF6 =	.913468973
M2OF6 =	.998761864
M1OF6 =	.99995353
MS6OF6 =	4.86161619*10↑ -10
MS4OF6 =	8.86300499*10↑ -6
MS2OF6 =	.0109400896
MS1 OF6 =	.156884438
TM1 OF2 =	.990043251
CM1 OF2 =	.994252508
R6OF6 =	.623105146
R5OF6 =	.929789461
RP =	.994252508
RP5OF6 =	.999512041
RP6OF6 =	.96600677
DA =	.90021649
L1TYP =	.0559715171
L2TYP =	.782841965
RATYP =	.894784999
L1DGR =	.0835199757
L2DGR =	.809994292
RADGR =	.977034243

REL:	3LU,6M	2LU,4M	1LU,2M	1LU,1M
FULL I/O	.0793	.5973	.8191	.8200
83% I/O	.0896	.6748	.9254	.9265
FULL I/O (W MS)	.1362-09	.5795-05	.8972-02	.1287+00
83% I/O (W MS)	.1539-09	.6547-05	.1014-01	.1454+00

TABLE A2.-- Continued

OUTPUT OF PROGRAM

SOLAR PROBE 12 MONTH RELIABILITY FOR FAILURE RATE= 1.0-05

CG=	.982632583
TC2OF3=	.980076116
LU3OF3=	.0252431613
LU2OF3=	.207670585
LU1OF3=	.647126436
M6OF6=	.0149237705
M4OF6=	.336630383
M2OF6=	.887010429
M1OF6=	.983646525
MS6OF6=	9.03515412*10↑ -32
MS4OF6=	3.02025525*10↑ -20
MS2OF6=	6.73072773*10↑ -10
MS1OF6=	4.01914096*10↑ -5
TM1OF2=	.912624347
CM1OF2=	.946590784
R6OF6=	.206635274
R5OF6=	.57928122
RP=	.946590784
RP5OF6=	.96295007
RP6OF6=	.719405299
DA=	.704406271
L1TYP=	.0430251993
L2TYP=	.356960416
RATYP=	.443010815
L1DGR=	.120616821
L2DGR=	.477804456
RADGR=	.719038099

REL:	3LU,6M	2LU,4M	1LU,2M	1LU,1M
FULL I/O	.0001	.0185	.1522	.1688
83% I/O	.0002	.0403	.3307	.3667
FULL I/O (W MS)	.6047-33	.1663-20	.1155-09	.6896-05
83% I/O (W MS)	.1314-32	.3613-20	.2509-09	.1498-04

TABLE A2.-- Continued

OUTPUT OF PROGRAM

SOLAR PROBE 24 MONTH RELIABILITY FOR FAILURE RATE= 1.0-05

CG=	.965566793
TC2OF3=	.930817737
LU3OF3=	6.37217192*10↑ -4
LU2OF3=	.0209405619
LU1OF3=	.236579128
M6OF6=	2.2271892?*10↑ -4
M4OF6=	.0356304994
M2OF6=	.457030994
M1OF6=	.816545917
MS6OF6=	8.163401*10↑ -63
MS4OF6=	6.08142487*10↑ -41
MS2OF6=	3.02028762*10↑ -20
MS1OF6=	2.69233919*10↑ -10
TM1OF2=	.746173665
CM1OF2=	.832882999
R6OF6=	.0426981365
R5OF6=	.219845897
RP=	.832882999
RP5OF6=	.735689382
RP6OF6=	.333813568
DA=	.496188194
L1TYP=	.0106739137
L2TYP=	.0821858099
RATYP=	.103533637
L1DGR=	.05495828
L2DGR=	.18112873
RADGR=	.29104529

REL:	3LU,6M	2LU,4M	1LU,2M	1LU,1M
FULL I/O	.0000	.0000	.0021	.0037
83% I/O	.0000	.0001	.0129	.0231
FULL I/O (W MS)	.1004-66	.2458-43	.1379-21	.1230-11
83% I/O (W MS)	.6221-66	.1523-42	.8546-21	.7618-11

TABLE A2.-- Continued

OUTPUT OF PROGRAM

```
      SOLAR PROBE 36 MONTH RELIABILITY FOR FAILURE RATE= 1.0-05

          CG=      .948797391
       TC2OF3=     .864458556
       LU3OF3=    1.60853763*10↑ -5
       LU2OF3=    1.87948082*10↑ -3
       LU1OF3=     .0738339177
        M6OF6=    3.32380616*10↑ -6
        M4OF6=    2.72103028*10↑ -3
        M2OF6=     .160317302
        M1OF6=     .542402511
       MS6OF6=    7.37575859*10↑ -94
       MS4OF6=    1.22451015*10↑ -61
       MS2OF6=    1.35527312*10↑ -30
       MS1OF6=    1.80351199*10↑ -15
       TM1OF2=     .576873266
       CM1OF2=     .702507782
        R6OF6=    8.82294115*10↑ -3
        R5OF6=     .0723414373
           RP=     .702507782
       RP5OF6=     .425610167
       RP6OF6=     .120200656
           DA=     .349518076

        L1TYP=    2.00594147*10↑ -3
        L2TYP=     .0146840589
        RATYP=     .0186959418
        L1DGR=     .0164472013
        L2DGR=     .0519937659
        RADGR=     .0848881684


 ERROR AT STEP      5.04
 NUMBER OUT OF RANGE
←SET MS6OF6 = 0
←GO
```

REL:	3LU,6M	2LU,4M	1LU,2M	1LU,1M
FULL I/O	.0000	.0000	.0000	.0000
83% I/O	.0000	.0000	.0001	.0005
FULL I/O (W MS)	.0000+01	.1719-66	.7475-34	.9947-19
83% I/O (W MS)	.0000+01	.2764-65	.1202-32	.1599-17

APPENDIX B

LOGIC DESIGN SIMULATION

This appendix will describe one of many different ways of simulating a logic design once the logic drawings exist. Many organizations performing digital logic design use simulators similar to the one described in this appendix. The particular one described has the primary advantages of ease of writing the simulation software and no requirement to input Boolean equations.

The simulator described here is a version of the present-day simulator used at the Applied Research Laboratory. Experience on this simulator has shown us that no logic design knowledge is necessary to produce the input information. Experience has shown that a secretary with less than one-half hour training can produce input cards from a logic drawing. However, an engineer is needed to debug any errors in the logic design or the input process.

B.1 General Description

To input a design, the user inputs equations of the form:

A = NAND(B,C,D)

XYZ = ANDNOR(A,B,C,D,AA,BB,CC,DD)

L = FFD(XYZ,ZZ,W,A,B,C,D,E,Ø,CP1)

with one equation per line or card. The first equation is a three-input NAND gate whose inputs are B,C,D, and output is A. The second equation represents a SUHL ANDNOR which has two 4-input AND gates feeding a NOR gate. The third equation represents a D-type flip-flop made by Transitron. The commas devide the various input gating levels of this flip-flop, the last field being the clock pulse input. The simulation clock pulses are represented by the letters CP followed by a one- or two-digit number. A normal logic level may appear at this position if the user does not want the flip-flop to be triggered by the simulation clock pulse. Note that the next to last field contains a zero instead of a name of a signal.

The above discussion may be clarified with an example. The logic diagram of Figure 37 represents a three-stage feedback shift register. The input to the first stage of this shift register is the exclusive OR of the second and third stage. The register shifts whenever the logic level SHIFT is a "1" and will have a repeating pattern of length 7. In case the register starts with all stages a zero, this state is decoded (ZEROA) and a 1 is fed to the first stage. In addition, the register may be loaded by setting LOAD to a 1 and the register will hold its current value when neither LOAD nor SHIFT is a 1.

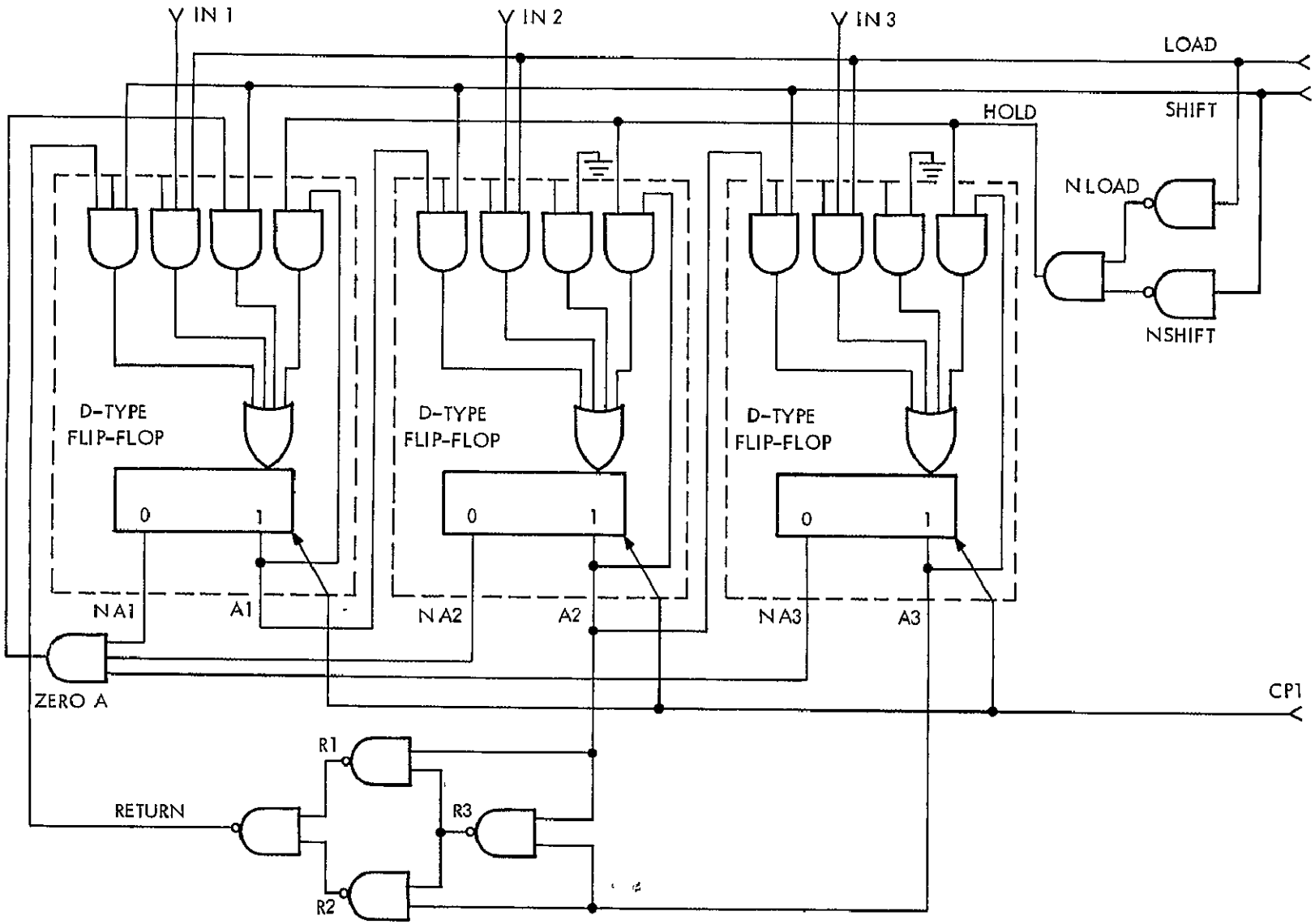


Figure 37. An Example of a Three-Stage Feedback Shift Register

This logic would be entered with the following equations:

```
A1 = FFD(RETURN,1,SHIFT,IN1,1,LOAD,ZEROA,SHIFT,HOLD,CP1)

A2 = FFD(A1,1,SHIFT,1,IN2,LOAD,1,Ø,HOLD,CP1)

A3 = FFD(A2,1,SHIFT,1,IN3,LOAD,1,Ø,HOLD,CP1)

HOLD = AND2(NLOAD,NSHIFT)

NLOAD = NAND1(LOAD)

NSHIFT = NAND1(SHIFT)

RETURN = NAND(R1,R2)

R1 = NAND2(A2,R3)

R2 = NAND2(A3,R3)

R3 = NAND2(A2,A3)

ZEROA = AND3(NA1,NA2,NA3)
```

B.2 Method of Simulation

Three general approaches to simulating MULTIPAC are possible. The best long-term solution would be to have a general purpose simulator written which would accept the equations as input. This then allows the simulator to be used on many projects, including the digital portions of the experiments surrounding MULTIPAC. A second solution is to use a general purpose simulator available on a commercial time-sharing service. As an example, a firm, RAPIDATA, supplies a terminal service which includes a digital logic simulator whose input format is of the general form shown here.

If neither of the first two methods of simulation are desirable, then a simulator for MULTIPAC only could be written to simulate the MULTIPAC design.

B.3 Writing a Simulator for MULTIPAC

The above format simplifies the writing of a simulator for a specific system to be simulated. It will be noted that the format of the logic is that of a Fortran statement where the output equals a function of a number of inputs. Thus, if a Fortran function is written for each of the required gates or flip-flops such that the proper Boolean function is performed and the Boolean answer is returned, then the statement could be entered as part of a Fortran program. The various size NAND gates need different function names since Fortran is incapable of accepting a varying number of arguments to a function.

Since Fortran functions only return a single value, then an additional equation would have to be added for every flip-flop. These equations will be for the zero output side of the flip-flop. For the example shown, equations of the following form would have to be added:

```
NA1 = INV(A1)

NA2 = INV(A2)

NA3 = INV(A3)
```

and the function INV would have to be written, which would simply invert the signal (i.e., ZEROS would become ONES and ONES would become ZEROS).

Thus, a Fortran function must be written for every different circuit type. For the above equations, functions must be written for FFD, INV, AND2, AND3, NAND1, and NAND2. Many will be very short functions. The NAND2 function will simply form the Boolean NAND of its two arguments by using Fortran IV Boolean functions, or a machine language routine. Each time the program containing the logic equations calls NAND2, it will return the NAND of the two inputs. For example, when the program executes the RETURN equation, it will perform the function NAND2 on the values of R1 and R2 at that time and set RETURN to the NAND of those values.

These equations would have to be surrounded by a Fortran program which creates clock pulses, prints outputs, and otherwise exercises the the design. The first part of the program would initialize all the signals, and set up a Fortran DO loop which contains the equations above as the main part of the loop. Each time the loop is entered, the clock pulses and the input signals would be varied according to some predetermined test pattern. On every exit from the loop the states of those signals traced would be printed on the line printer. The printout would be of the form shown below.

A1	A2	A3	RETURN	ZEROA
Ø	Ø	Ø	Ø	1
1	Ø	Ø	Ø	Ø
Ø	1	Ø	1	Ø
1	Ø	1	1	Ø
1	1	Ø	1	Ø
1	1	1	Ø	Ø
Ø	1	1	Ø	Ø
Ø	Ø	1	1	Ø
1	Ø	Ø	Ø	Ø

Most logic equations will produce a single bit of information (Ø or 1). Allowance could be made for multiple bit operations. For example, a new function, REG, could be written to form a "register" of signals for purposes of printout. If the equation

Z = REG(A1, A2, A3)

were included and Z traced instead of A1, A2 and A3, the printout of the above example would be:

Z	RETURN	ZEROA
Ø	Ø	1
4	Ø	Ø
2	1	Ø
5	1	Ø
6	1	Ø
7	Ø	Ø
3	Ø	Ø
1	1	Ø
4	Ø	Ø

where A1, A2, A3 are now the three bits of "register" Z.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY *(Corporate author)*: Applied Res. Lab., Sylvania Electronic Systems, An Operating Group of Sylvania Electric Products Inc. 40 Sylvan Road, Waltham, Mass. 02154

2a. REPORT SECURITY CLASSIFICATION: Unclassified

2b GROUP: N/A

3 REPORT TITLE: MULTIPAC, A Multiple Pool Processor and Computer for a Spacecraft Central Data System

4 DESCRIPTIVE NOTES *(Type of report and inclusive dates)*: Final Report Phase II

5 AUTHOR(S) *(First name, middle initial, last name)*: Thomas E. Baker; Robert L. South; Gene A. Cummings;

6 REPORT DATE	7a TOTAL NO. OF PAGES	7b NO. OF REFS
October 1969	215	3

8a CONTRACT OR GRANT NO	9a ORIGINATOR'S REPORT NUMBER(S)
NAS2-3255	F-7159-1
b PROJECT, TASK, AND WORK UNIT NO	
c DOD ELEMENT	9b OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d DOD SUBELEMENT	NASA CR-73348

10 DISTRIBUTION STATEMENT: Distribution of this report is provided in the interest of information exchange. Responsibility for the contents resides in the author or organization that prepared it.

11 SUPPLEMENTARY NOTES	12 SPONSORING MILITARY ACTIVITY
	National Aeronautics and Space Administration, Ames Res. Center, Moffett Fld., Calif.

13 ABSTRACT

This report contains a detailed description of a large-scale integrated circuit version of a central data system for deep space probes which is made up of pools of identical modules which are interconnected by programs to form one or more computers. These modules are then reconfigured after a module failure by reprogramming via the command and telemetry links.

DD FORM 1 NOV 65 1473

UNCLASSIFIED
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer Logic						
Computers						
Data Processing Systems						
Digital Computers						
Special Purpose Computers						
Space Probes						
Spacecraft						